Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

David Novak

# Knjižnica za analizo besedilnih dokumentov v programskem okolju Orange

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: prof. dr. Blaž Zupan

Ljubljana, 2016

# Izjava o avtorstvu zaključnega dela

Spodaj podpisani David Novak, vpisna številka 63090102, avtor zaključnega dela z naslovom:

*Knjižnica za tekstovno analizo v programskem okolju Orange* (angl. *Text mining library for Orange data mining suite*)

IZJAVLJAM

1. da sem pisno zaključno delo študija izdelal samostojno pod mentorstvom prof. dr. Blaža Zupana;
2. da je tiskana oblika pisnega zaključnega dela študija istovetna elektronski obliki pisnega zaključnega dela študija;
3. da sem pridobil/-a vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v pisnem zaključnem delu študija in jih v pisnem zaključnem delu študija jasno označil/-a;
4. da sem pri pripravi pisnega zaključnega dela študija ravnal/-a v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil/-a soglasje etične komisije;
5. soglašam, da se elektronska oblika pisnega zaključnega dela študija uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;
6. da na UL neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve avtorskega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja pisnega zaključnega dela študija na voljo javnosti na svetovnem spletu preko Repozitorija UL;
7. dovoljujem objavo svojih osebnih podatkov, ki so navedeni v pisnem zaključnem delu študija in tej izjavi, skupaj z objavo pisnega zaključnega dela študija.

V Ljubljani, dne 18. junija 2016 Podpis študenta/-ke:

*Zahvaljujem se mentorju prof. dr. Blažu Zupanu za strokovno vodenje in napotke pri ustvarjanju in pisanju magistrskega dela.*

*Zahvaljujem se Niku Colneriču in Jerneju Kerncu, za pomoč pri snovanju dela in za njune prispevke h končnemu izdelku.*

*Zahvaljujem se tudi vsem ostalim članom Laboratorija za bioinformatiko, ki so na kakršenkoli način pripomogli k uresničitvi te magistrske naloge.*

*Zahvaljujem se lektorici, Urški Rupar Vrbinc, za strokovno pomoč pri lektoriranju magistrskega dela.*

*Zahvaljujem se staršem, družini in prijateljem za vso podporo in razumevanje, ki so mi ga tekom mojega študija namenili.*

# Kazalo

# Povzetek

Razvili smo sistem za analizo besedil in ga osnovali kot dodatek za programsko okolje Orange. Orange združuje bogat nabor metod za nadzorovano in nenadzorovano strojno učenje, zato je odličen temelj za razvoj takega sistema. S pregledom literature in odprtih orodij smo določili kaj so temeljne metode, ki se uporabljajo na tem področju in na podlagi le-tega osnovali funkcionalnosti naše knjižnice. Dodali smo gradnike za zajem podatkov s spletnih virov kot sta PubMed in New York Times. Implementirali smo metode za predobdelavo, ki vključujejo pretvorbo besedil v vektorje, odstranjevanje odvečnih besed, lematizacijo in krnjenje, tok dela pa nato podprli z vizualizacijami, na primer z oblakom besed. Naš cilj je bil razviti gradnike, ki se med seboj dobro povezujejo z vizualnim programiranjem, so dobro povezljivi z ostalimi gradniki sistema Orange, ter jih je moč enostavno nadgraditi z razvojem novih gradnikov.

**Ključne besede:** analiza besedil, predobdelava podatkov, vizualizacija, vizualno programiranje.

# Abstract

We have developed a text mining system that can be used as an add-on for Orange, a data mining platform. Orange envelops a set of supervised and unsupervised machine learning methods that benefit a typical text mining platform and therefore offers an excellent foundation for development. We have studied the field of text mining and reviewed several open-source toolkits to define its base components. We have included widgets that enable retrieval of data from remote repositories, such as PubMed and New York Times. The pre-processing was designed to include transformation of documents to vectors, stop word removal, lemmatization and stemming. The results can be visualized via widgets such as the word cloud. Our goal was to develop widgets that can be easily incorporated into the existing Orange workflow, can be upgraded with additional widgets, and perform well in a visual programming environment.

**Keywords:** text mining, data pre-processing, visualization, visual programming.

# Poglavje 1

# Uvod

Eden izmed temeljev moderne informacijske družbe je razumevanje podatkov, ki jih ustvarja. Čeprav ti v obtok vstopajo prek mnogih medijskih kanalov, se za formalno izmenjavo informacij večinoma še vedno uporablja pisana beseda [1]. Nizka cena in preprostost digitalnega shranjevanja sta arhiviranje dokumentov naredili trivialno. S kopičenjem, pa raste tudi količina neobdelanih podatkov, ki lahko potencialno vsebuje veliko koristnih informacij. Kakovostna analiza besedil (angl. *text mining*) zato še nikoli ni bila tako pomembna kot je danes [2]. S programsko realizacijo analize besedil pridemo do vrste programskih rešitev, ki združujejo metode statistike, strojnega učenja in obdelave naravnega jezika. Te nato uporabimo kot orodje za reševanje problema pridobivanja znanj.

Cilj te magistrske naloge je raziskati področje analize besedil, določiti, kaj so temeljne operacije, ter s tem znanjem razviti programsko knjižnico, ki bo služila analizi besedil. Kot temelj za razvoj smo uporabili Orange, programsko okolje ustvarjeno za rudarjenje podatkov (angl. *data mining*), ki ga razvija Laboratorij za bioinformatiko Univerze v Ljubljani. Orange je odlična podlaga za tak sistem, saj združuje bogat nabor že razvitih metod za nadzorovano in nenadzorovano učenje, ki smo jih lahko uporabili pri uvrščanju besedil oziroma smo jih povezali s komponentami, ki smo jih zasnovali v našem delu. Orange verzije 2 že vsebuje dodatek za analizo besedil,

vendar smo ga za verzijo 3 želeli prenoviti in dopolniti. Pri razvoju smo se osredotočili na učinkovito integracijo in poskrbeli, da so novi gradniki dobro vpeti v že obstoječ tok dela. Povezljivost med komponentami in enostavnost uporabe Orange rešuje s podporo za vizualno programiranje, ki smo ga s pridom uporabili tudi v naši rešitvi.

Pri izboru in implementaciji tehnik smo se zgledovali po opisih metod v delih [2, 3]. V knjižnici smo podprli zajem, predobdelavo in vizualizacijo besedilnih podatkov. Zajem vključuje pridobivanje besedil s spletnih virov PubMed in New York Times, omogoča pa tudi uporabo lokalno shranjenih dokumentov. V proces predobdelave smo dodali razčlenjevanje, odstranjevanje odvečnih besed, krnjenje in lematizacijo. Modul omogoča tudi postavitev zgornje in spodnje meje za izključevanje besed na podlagi njihovih frekvenc v dokumentih. Med vizualizacije smo vključili pregledovalnik korpusov, oblak besed in zemljevid geografskih entitet. V tok dela je vključeno tudi prepoznavanje tem z LDA (latentna Dirichletova alokacija) in bogatenje besed.

Nekateri elementi analize, kot so metode za prevedbo besed na leme ali korene, so vse prej kot trivialni. Gradnike in funkcionalnosti smo zato realizirali z uporabo elementov knjižnice *Natural Language Toolkit* (v nadaljevanju NLTK), ki podpira več jezikov in vsebuje generične baze podatkov, kot je baza najpogostejših besed. Vsebuje tudi bogat nabor podatkov za analizo, kot je Brownov korpus. Orange in njegovi dodatki so napisani v programskem jeziku Python verzije 3, tega pa smo se držali tudi pri implementaciji naše knjižnice.

Pričujoče magistrsko delo skupaj z uvodom šteje šest poglavij. V drugem poglavju opišemo področje analize besedil, skupaj s kratkim pregledom njegove zgodovine in razvoja. Definiramo njegov izvor iz podatkovnega rudarjenja in navedemo, v čem sta si ti veji podobni, in kje najdemo razlike, zaradi katerih jih prepoznamo kot ločeni področji. Del poglavja namenimo tudi temeljnim operacijam, ki so potrebne za uspešno analizo, za zaključek pa vključimo še pregled že obstoječih programskih rešitev, ki prav tako rešujejo problem pridobivanja znanj iz besedil.

Tretje poglavje je namenjeno tehničnim podrobnostim magistrske naloge in opisuje platformo Orange, uporabljene programske knjižnice in predstavi kaj za našo implementacijo pomeni uporaba programskega jezika Python.

V četrtem poglavju naštejemo funkcionalnosti knjižnice, predstavimo njeno arhitekturo in razložimo kako smo temeljne operacije analize besedil predstavili z Orange gradniki in zakaj smo se tako odločili.

To v petem poglavju podpremo s slikami shem v Orange in primeri uporabe. Za zaključek ocenimo končni produkt, ga primerjamo z že obstoječimi orodji in izpostavimo njegove prednosti oziroma identificiramo, kje je prostor za nadaljne izboljšave.

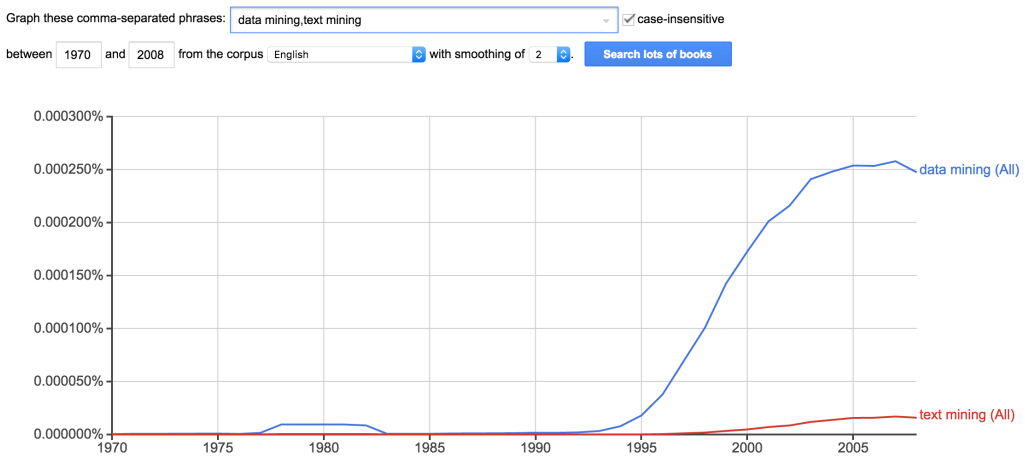
# Poglavje 2

# Analiza besedil

Analiza besedil je interdisciplinarno področje, ki iz besedil z uporabo statistike, strojnega učenja in obdelave naravnega jezika odkriva nova znanja [3]. Izvira iz podatkovnega rudarjenja, vendar je v primerjavi z njim še relativno mlado in neuveljavljeno področje. V zadnjem desetletju je šele zares vzbudilo zanimanje ter doživelo temu primeren skok v razvoju. To je morda nenavadno, kajti prepoznavanje vzorcev v podatkih ni nekaj novega. Bayesov teorem (Thomas Bayes) in regresijska analiza (Adrien-Marie Legendre) sta dva zelo pomembna temelja tega koncepta, od tega je bil prvi predstavljen že leta 1763 [4]. Podatkovno rudarjenje, ki temelji na njima, je veja računalništva z aplikacijami na mnogih področjih. Z dovolj veliko bazo podatkov o potrošniških navadah kupcev smo sposobni napovedati kateri produkti so za njih najbolj relevantni. Gledalcem lahko na podlagi filmov, ki so si jih že ogledali, ponudimo nove za katere so naši napovedni modeli predvideli, da jim bodo všeč. Kje torej tiči vzrok za tako pozen razvoj analize besedil, sploh če je res tako sorodna podatkovnemu rudarjenju?

Za boljšo predstavo o prisotnosti teh znanosti skozi čas smo preučili zanimanje za njih. To najlažje storimo z merjenjem pojavitev ključnih besed v objavljenih delih. Spletni brskalnik Google ponuja orodje[1], ki v izbranem korpusu poišče frekvence pojavitev besednih zvez, ki mu jih podamo. Za to

[1] `https://books.google.com/ngrams`

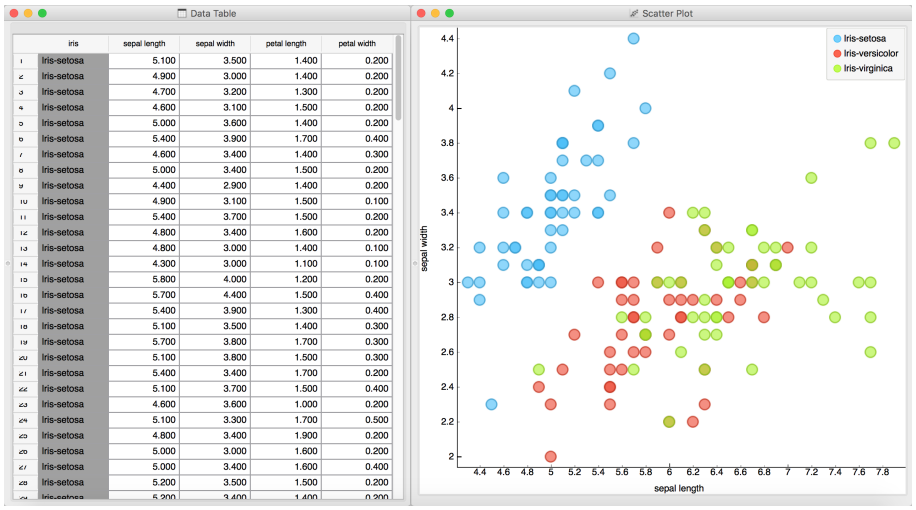



Slika 2.1: Na sliki vidimo graf generiran z orodjem Google Ngram Viewer. Graf prikazuje frekvenco pojavitev besednih zvez *"data mining"* in *"text mining"* v angleški literaturi od leta 1970 do leta 2008. Po letu 1993 je razlika med krivuljama vedno bolj izrazita.

dokumente predstavi z $n$-terkami znakov in prešteje njihove pojavitve. Podrobnosti o tem kako deluje predstavitveni model z $n$-terkami bomo opisali v nadaljevanju. Zanimalo nas je kaj nam taka predstavitev lahko pove o prisotnosti tem podatkovnega rudarjenja oziroma analize besedil v zgodovini pisane besede. V angleški literaturi od leta 1970 do leta 2008[2], smo poiskali pojavitve besednih zvez "*data mining*" in "*text mining*" in dobili zanimive rezultate. Če sodimo po grafu, ki ga orodje izriše (glej sliko 2.1) lahko prve znatne pojavitve besedne zveze "*data mining*" zasledimo že leta 1977. Nekoliko pogosto jih opazimo po letu 1993, ko postaja razlika med krivuljama vedno bolj očitna. Porast literature o *"text mining"* zasledimo šele po letu 1998. Pri tem moramo upoštevati, da so pojavitve zanemarjene, če se besedna zveza ne pojavi v vsaj 40 delih. Leta 2008 se rudarjenje podatkov omeni približno petnajstkrat tolikokrat kot analiza besedil. Vzrok temu so lahko tudi začetna nesoglasja glede poimenovanja stroke, še vedno pa lahko zaključimo, da je rudarjenje podatkov kot področje bolj uveljavljeno.

Odgovor, ki utemelji to razliko, se skriva v značilnostih obravnavanih

[2]V orodju je 2008 najkasnejše leto, ki smo ga lahko izbrali.




Slika 2.2: Na sliki vidimo podatkovno množico rože perunike prikazano z gradniki v Orange. Na levem delu slike je množica prikazano s tabelo, na desnem pa z grafom raztrosa. Na slednjem je dolžina čašnih listov prikazana na x, njihova širina pa na y osi. Razred, kateremu primer pripada je izražen z barvo, ki jo lahko preberemo iz legende.

podatkov. Podatkovno rudarjenje se ukvarja s prepoznavanjem vzorcev iz baz podatkov organiziranih v neko dogovorjeno strukturo. Kot primer tega si lahko ogledamo podatkovno množico rože perunike (angl. *iris data set*, primer množice je na sliki 2.2). To je angleški statistik Fisher leta 1936 predstavil v svojem delu o linearni diskriminantni analizi [5], v demonstrativne namene pa se uporablja še danes. Recimo da želimo izvedeti, kako lahko na podlagi dimenzij venčnih in čašnih listov napovemo podvrsto specifične rože. Za vsak zabeležen primerek rože se v množici nahaja en vnos, ki ga sestavljajo štiri številske vrednosti: dolžina in širina za obe vrsti listov. Podatki sledijo nekemu naboru pravil, ki ga imenujemo podatkovni model. Ta narekuje njihovo zgradbo in pripadajoče medsebojne povezave. Take podatke smatramo kot *strukturirane*. Za človeka množica števil na prvi pogled morda ni pretirano informativna kar pa niti ni čudno, saj je oblika v kakršni so ti podatki namenjena algoritmični obdelavi. Res je, da vsebuje znanja, vendar so ta implicitna.

V nasprotju s tem imamo na drugi strani analizo besedil, kjer so vhodni podatki besedila. Ta so ustvarjena za branje in so zato že po definiciji človeku razumljiva. Za ekstrakcijo znanj človeku ni potrebno izvajati analize, dovolj je, da besedilo prebere. Ta pristop je dovolj dober, če moramo prebrati knjigo ali dve. Če pa želimo analizirati nekaj deset tisoč Twitter objav in iz njih prepoznati odzive javnosti, pa ta pristop ni primeren. Logično je, da bi zato želeli tak proces avtomatizirati, kar pa otežuje dejstvo, da besedila vsebujejo veliko različnih vrst podatkov, kot so števila in datumi. Teh računalnik v poplavi znakov ne zna identificirati kot bistvenih, kaj šele postaviti v ustrezen kontekst. Človeku je razumevanje in uvrščanje besedil trivialno, računalnikov ki bi znali to početi na enak način, pa še dolgo ne bomo imeli na voljo. Dokler besedil ne prevedemo na nek podatkovni model, jih obravnavamo kot *nestrukturirane*.

Težava je torej v obdelovanju nestrukturiranih podatkov, kar pa je že dokaj star problem. Prvi odločilni sistemi so bili zato ustvarjeni za delo s številskimi podatki, čeprav je bil njihov prvotni fokus na besedilih [6]. Tak pristop je namreč bolj preprost in ne vključuje dela s korpusi besedil, ki znajo biti za obdelavo težavni. Poslovna inteligenca (angl. *business intelligence*) je prav tako eno izmed področjih, ki jih najdemo v tesni povezavi z analizo besedil [6].

Proces predobdelave je temeljnega pomena, če želimo besedila analizirati programsko. Ta v grobem vključuje razčlenjevanje besedil na manjše elemente, izločanje odvečnih elementov in njihovo transformacijo s pomočjo krnjenja ali lematizacije. Osnovna entiteta nad katero izvajamo tekstovno analizo, je *dokument*. V vsakdanjem kontekstu pojem dokument opisuje uradno listino ali obrazec, v okviru analize besedil pa je njegov pomen bolj abstrakten. Navezuje se na poljubno telo besedila, ki ga lahko (vendar ne nujno) povežemo z nekim realnim dokumentom [3]. Običajno na vhod ne podamo enega samega dokumenta, ker iz tako majhnega vzorca težko izvlečemo kaj uporabnega. Po navadi podamo zbirko dokumentov imenovano tudi *korpus*. Elementi korpusa so lahko tematsko povezani, ni pa nujno.

## 2.1 Temeljne operacije in postopki

V tem poglavju so zbrane in opisane metode, ki predstavljajo temelje tipičnega sistema za analizo besedil. Kako so združeni v celoto, je problem, ki ga rešuje vsaka platforma po svoje. Odvisen je namreč od izbranih tehnologij in vizije, ki jo imamo o končnem produktu. Programsko okolje, kot je Orange, je podprto na vseh večjih operacijskih sistemih in svoje procese izvaja lokalno, eno izmed orodij, ki smo jih preizkusili, pa svoje procese izvaja na oblaku. Spet neko drugo orodje uporabniku ponuja obe možnosti.

### 2.1.1 Zajem podatkov

Vsak analitični proces se začne z zajemom podatkov. Danes je ta korak z uporabo interneta precej trivialen. Repozitorijev z besedili je na spletu ogromno, značilno je tudi, da podpirajo iskanje in filtriranje po njihovi vsebini. Izkaže se, da je to precej nujno, saj analize običajno izvajamo nad zbirkami velikostnega reda nekaj tisoč do celo nekaj milijonov dokumentov. V tem pogledu lahko ločimo *statične* in *dinamične* zbirke dokumentov [3]. Razvrstitev je odvisna od tega, ali se zbirka posodablja z novimi vnosi, ali pa jo sestavlja neka ustaljena množica elementov. Statični so na primer leposlovni korpusi, sploh če je njihova vsebina omejena z nekim časovnim intervalom. Primer dinamične zbirke je strokovna knjižnica biomedicinskih del PubMed[3], ki je v našem dodatku predstavljena kot eden izmed gradnikov. PubMed danes vsebuje več kot 26 milijonov člankov, vsak dan pa se baza dopolni z nekaj deset tisoč vnosi. Pomemben vir informacij, ki smo ga vključili tudi v naš dodatek, so svetovne novice[4]. Novičarski portali se posodabljajo večkrat dnevno, vsebina objav pa nam lahko pove veliko o razvoju trenutnih trendov.

[3] `http://www.ncbi.nlm.nih.gov/pubmed`
[4] `https://developer.nytimes.com/`

### 2.1.2 Predobelava podatkov

Omenili smo že, da besedila uvrščamo med nestrukturirane podatke, ker nimajo dogovorjene ureditve. Ta trditev sicer popolnoma ne drži, ker je pojem strukturiranosti odvisen od konteksta v katerem ga obravnavamo [3]. Z vidika jezikoslovja ima vsako besedilo nekakšno (lahko tudi čisto preprosto) zgradbo. Presledki in ločila ločujejo posamezne besede, skupine teh so nato združene v odstavke, ti pa v poglavja. Brez osnovne zgradbe bi bilo besedilo za človeka težko berljivo ali celo neberljivo. Omenili smo tudi, da nam tako strukturiranje ne pomaga, če želimo besedila analizirati programsko, zato je besedilo potrebno najprej ustrezno obdelati. Cilj pri tem je določiti atribute, s katerimi lahko besedilo učinkovito predstavimo. Kompleksnost predobdelave v praksi prilagodimo vhodnim podatkom in pričakovanjem, ki jih imamo glede končnih rezultatov. Zapleteni procesi ustvarijo boljše modele za učenje, vendar so časovno potratni, kar pride do izraza pri obsežnih korpusih. V nadaljevanju opisujemo modele predstavitve besedil, kaj so odvečne besede, oziroma kako le-te odstranimo in transformacijo besed.

#### Predstavitev besedilnih dokumentov

Predobdelavo besedila začnemo tako, da ga razbijemo na manjše obvladljive elemente imenovane žetone (angl. *tokens*). Ta proces imenujemo razčlenjevanje (angl. *tokenization*), elementi razbitja pa so v praksi fraze ali pa posamezne besede. Členitev je temelj za grajenje različnih predstavitvenih modelov:

**Vreča besed** Najpogostejši in morda tudi najbolj intuitiven način predstavitve besedil je z vrečo besed (angl. *bag of words*), ki jo sestavljata množica besed (atributov) in njihove frekvence. Posamezne besede najlažje izluščimo s členitvijo besedila po presledkih. V večini primerov je to dovolj, saj se med sosednjimi besedami pogosto nahaja vsaj en presledek, v nasprotnem primeru, pa se poslužimo tudi členitve po ločilih. Pri tem ne smemo upoštevati takih, ki se lahko pojavijo kot del besede (na primer opuščaj ali vezaj), lahko

pa ta problem rešimo tako, da obliko besed definiramo z regularnimi izrazi. To nam lahko vrne dobre rezultate, vendar je zato tudi temu primerno zamudno.

Pri štetju pojavitev je značilno, da vse velike črke v besedah pretvorimo v male. Če bi ohranili velike začetnice in ostale velike črke, bi enake besede (po možnosti tudi z istim pomenom) obravnavali kot različne, kar bi otežilo štetje njihovih pojavitev. Pojavitve lahko predstavimo s celimi števili, še bolje pa je, če jih predstavimo z relativnimi frekvencami. Pretvorba števila pojavitev v relativne frekvence je način normalizacije. Velikokrat nas ne zanima kolikokrat se določena beseda pojavi. Bolj pogosto nas zanima kakšen je odstotek njene prisotnosti v besedilu, oziroma kakšna je verjetnost, da bi izmed vseh besed naključno izbrali prav to. Pojavitve besed lahko definiramo tudi binarno, kar pomeni, da za to uporabimo dve vrednosti: ali se beseda pojavi, ali pa ne. Taka predstavitev se uporablja pri definiranju mer, kot je podobnost po Jaccardu.

Vreča besed je model vektorskega prostora, ki besedilo predstavi z visokodimenzionalnim vektorjem. Z združitvijo vektorjev večih dokumentov ustvarimo matrično predstavitev korpusa, kjer stolpci ustrezajo besedam, vrstice pa dokumentom. Kljub temu, da so si dokumenti istega korpusa lahko tematsko sorodni, se v matrikah obsežnih korpusov hitro pojavi ogromno stolpcev, ki so za večino dokumentov "prazni", oziroma z drugimi besedami frekvence vsebovanih besed so enake nič. Uporaba redkih matrik (angl. *sparse matrix*) je na tem področju zato pogosta, saj znatno olajša shranjevanje takih modelov. Ena izmed glavnih pomankljivosti vreče besed je zanemarjanje vrstnega reda besed, ter posledično tudi konteksta, v katerem se nahajajo. Brez tega za besede z enako sintakso težko določimo pripadajočo semantično vrednost, izgubimo pa tudi logične povezave med njimi.

**TF-IDF** *Term frequency - inverse document frequency* je statistična mera, ki se uporablja za uteževanje atributov v predstavitvenih modelih. Z njo ocenimo pomembnost besede oziroma izraza za posamezen dokument. Če

izraz najdemo v malo dokumentih, lahko sklepamo, da je ta za njih specifičen in jih zato tudi dobro opiše. Po drugi strani izraz, ki se pojavi v večini dokumentov, ne pripomore k razlikovanju med njimi. Preden si pogledamo prednosti in slabosti tega uteževanja, najprej predstavimo pojme, s katerimi je mera definirana. Zapišimo frekvenco izraza (tf) v določenem dokumentu kot funkcijo $\mathrm{tf}(t, d)$, kjer $t$ predstavlja ta izraz, $d$ pa dokument. Frekvenco dobimo tako, da število pojavitev $t$ preprosto delimo s številom vseh besed v dokumentu. V krajših dokumentih je število pojavitev nekega elementa lahko manjše kot v daljših, zgolj na račun števila vsebovanih besed. Zato v ta namen raje uporabljamo frekvence.

Še vedno pa moramo tistim elementom, ki se pojavijo v veliko dokumentih, utež ustrezno zmanjšati. V ta namen definiramo frekvenco pojavitve v dokumentih (df). Ta predstavlja število dokumentov v katerih se izraz $t$ pojavi. Iz nje lahko izračunamo inverzno frekvenco pojavitve v dokumentih (idf):

$$\mathrm{idf}(t) = \log \frac{|D|}{|\{d \in D : t \in d\}|} = \log \frac{|D|}{\mathrm{df}(t)} \tag{2.1}$$

$D$ je množica vseh dokumentov, idf pa je naravni logaritem koeficienta števila vseh dokumentov in df. Elementi, ki se prek celotnega korpusa pojavijo velikokrat, imajo nizko vrednost idf. Če bi analizirali članke o analizi besedil, bi bila beseda *"dokument"* gotovo ena izmed teh. Končno vrednost TF-IDF uteži nekega izraza dobimo tako, da zmnožimo vrednosti tf in idf:

$$\mathrm{tfidf}(t, d) = \mathrm{tf}(t, d) * \mathrm{idf}(t) \tag{2.2}$$

S TF-IDF lahko določimo splošno pomembnost elementa v modelu, še vedno pa ne moremo določiti pomena elementov znotraj konteksta, v katerem se nahajajo.

**$n$-terke znakov** Skupine $n$ zaporednih znakov, imenujemo $n$-terke (angl. *$n$-grams*). Pri $n = 2$ lahko besedo "analiza" predstavimo s sledečim naborom dvojk ($n$-terk dolžine dva): "an", "na", "al", "'li", "iz" in "za". Skupine

znakov generiramo po principu drsečega okna, kjer na vsakem koraku okno dolžine $n$ premaknemo za en znak naprej in zabeležimo novo terko. Pred generiranjem terk besedilo običajno do neke mere predobdelamo. Neobdelana besedila so polna odvečnih znakov, ki bi jih tako posledično prenesli tudi na terke. Pri izbiri števila znakov moramo upoštevati, da z njim raste tudi število različnih[5] terk. Takih ki so sestavljene iz dveh znakov je bistveno manj kot takih sestavljenih iz treh ali več znakov, res pa je, da daljše terke omogočajo kompleksnejše analize. V praksi se uporabljajo vrednosti do $n = 5$, pri večjih dolžinah pa je generiranih elementov za učinkovito obdelavo že preveč. Predstavitev s terkami je lahko zelo informativna in ima veliko načinov uporabe. Iz nje lahko sestavimo dober model za prepoznavanje jezika, v katerem je besedilo napisano.

**Fraze** Podobno kot lahko združimo $n$ zaporednih znakov, lahko združimo tudi $n$ zaporednih besed in s tem ustvarimo *frazo*. Fraze so sestavljene iz zaporednih besed in zato omogočajo vpogled v kontekst besedila. Posamezne fraze lahko povežemo s temami, v katerih se najpogosteje pojavljajo, vendar procesiranje pri večjih frazah postane zelo časovno potratno.

### Krnjenje in lematizacija

Ko besedilo razčlenimo na posamezne besede, lahko med nekaterimi hitro opazimo podobnosti. Za primer vzemimo besedi *"hoditi"* in *"hodil"*, ki bi jih pravzaprav lahko obravnavali kot enaki. Izven konteksta besedila je njun informacijski doprinos skoraj enak. Problem morfološke zgradbe besed v procesu analize besedil rešujemo s krnjenjem in lematizacijo. Krnjenje je postopek odstranjevanja besednih končnic, s katerim pridobimo krn besede (za besedo *"hoditi"* bi bil to *"hodi"*). Na ta način se pri gradnji vektorskih modelov izognemo nepotrebnemu razlikovanju sicer sorodnih besed. Kolikšen del besede je potrebno odstraniti, algoritmi za krnjenje določajo na podlagi nabora pravil.

[5] Večkratne pojavitve enakih terk upoštevamo samo enkrat.

Lematizacija je nekoliko bolj zapleten postopek, ki besedam določi njihove osnovne oblike, imenovane leme. Lema je v veliki meri odvisna od besedne vrste, za katero jo želimo določiti. Glagol *"pišem"* ima lemo *"pisati"*, ki je pravzaprav njegova nedoločna oblika. Logika lematizacije je prav tako lahko definirana z naborom pravil, lahko pa algoritem razvijemo s strojnim učenjem. Lematizacija je lahko zelo učinkovita, vendar se zaradi hitrosti v procesih raje uporabljajo algoritmi za krnjenje (na primer *Porter*).

### Odstranjevanje neinformativnih besed

Manj pomembne besede (angl. *stop words*) so definirane kot najpogostejše besede v jeziku. V slovenščini jih po navadi najdemo med predlogi (na primer "k"-"h" in "s"-"z"), vezniki (na primer "in" in "ter") in členki (na primer "že" in "le"). Manj pomembne besede iz predstavitvenih modelov odstranjujemo zaradi pomanjkanja semantične vrednosti. S tem izboljšamo kakovost rezultatov in zmanjšamo dimenzije predstavitvenih modelov kar pohitri proces obdelave.

Manj pomembne besede jezika pa včasih niso edini atributi, ki jih želimo odstraniti. Včasih želimo odstraniti besede na podlagi minimalne ali maksimalne frekvence v dokumentih (angl. *minimum/maximum document frequency*). Če velja $min\_df = 100$, potem želimo iz množice odstraniti vse besede, ki se pojavijo v manj kot sto dokumentih. Podobno pri $max\_df = 1000$, obdržimo vse besede, ki se pojavijo v manj kot tisoč dokumentih. Meji intervala izražamo s celimi števili ali odstotkovno. Atribute lahko odstranimo tudi glede na njihovo dolžino in pripadnost besedni vrsti. Proces odstranjevanja vedno prilagodimo dokumentom, ki jih obdelujemo. Zakaj je to pomembno, si poglejmo na primeru znanega stavka iz Shakespearove tragedije Hamlet: "*To be or not to be*". Ta je v celoti sestavljen iz besed, ki jih sicer najdemo na seznamu manj pomembnih besed angleškega jezika. Če bi jih odstranili, bi s tem izgubili del informacije, ki pa je zaradi narave besedila pomembna.

### 2.1.3 Označevanje besednih vrst

Omenili smo že, da razumevanje naravnega jezika za računalnik ni trivialna naloga. V primerjavi z umetnimi jeziki se pri obdelavi naravnega jezika srečujemo z dvoumnostjo pomena besed, saj je ta namreč zelo pogosto odvisen od njenega konteksta. V slovenščini imajo besede lahko različen pomen, a enako sintaktično strukturo. Tako nam preostane ločevanje po fonetičnem zapisu, ali pa, kot že omenjeno, po besedah v njeni okolici. V angleščini je označevanje besednih vrst (angl. *part of speech tagging*) še nekoliko bolj zapleteno, saj lahko ena beseda pripada več različnim besednim vrstam. Vzemimo za primer besedo "*above*". V nekaterih primerih je lahko uporabljena kot samostalnik, spet v nekih drugih pa kot predlog. Določanje pripadnost besednim vrstam se tudi za človeka kdaj izkaže kot izziv. Kako torej k temu problemu pristopimo algoritmično?

Eden izmed praktičnih pristopov je *stohastično označevanje* [7]. Ta pristop temelji na verjetnosti pojavitve določene besedne vrste. Za delovanje potrebuje referenčno množico že označenih besedil. To so ponavadi referenčni korpusi kot je na primer Brownov korpus, ki vsebuje približno milijon označenih angleških besed iz različnih virov[6]. Označevanje s stohastičnimi metodami variira v kompleksnosti. Besede v besedilu lahko uvrstimo tudi tako, da preprosto pogledamo, kaj je s statističnega vidika, njena najpogostejša oznaka. Nekoliko bolj napredni so pristopi z uporabo Viterbi algoritma in skritih modelov Markova, kjer sicer ne vidimo stanj, imamo pa vpogled v izide, ki so v tem primeru posamezne besede [8].

### 2.1.4 Ocenjevanje podobnosti

Ko dokumente preslikamo v vektorski prostor, lahko nad to predstavitvijo uporabimo metode za razvrščanje. Podobnost med dokumenti lahko ocenimo z računanjem razdalj med vektorji, ki jih predstavljajo. Omenili smo že, da za beleženje pogostosti elementov raje uporabljamo frekvence, ker to

[6] `http://clu.uni.no/icame/manuals/BROWN/INDEX.HTM`

odpravi razliko v dolžini besedil. Temu primerno moramo upoštevati, da pri primerjanju takih vektorjev razlike bolj opiše vmesni kot, kot pa razlika v njihovi dolžini. Zaradi tega mere kot je Evklidska razdalja tu ne pridejo v poštev. Dovolj velika razlika v dolžini vektorjev, bi namreč popolnoma zasenčila informacije, ki jih podajata smeri.

### Kompresija

Izid kompresije dokumentov je lahko zgovorna mera za določanje podobnosti [9]. Tudi pri tej metodi oceno podamo na podlagi neke številske vrednosti, ki jo določa uspešnost kompresije. Preden razložimo kaj to pomeni, zapišimo kako bi z uporabo te metode izračunali razdaljo (mera je podobna tisti, ki jo najdemo v [10]):

$$\mathrm{D}(a,b) = \frac{\mathrm{Z}(a+b) - \mathrm{Z}(a)}{\mathrm{Z}(a)} + \frac{\mathrm{Z}(b+a) - \mathrm{Z}(b)}{\mathrm{Z}(b)} \tag{2.3}$$

V enačbi $a$ in $b$ predstavljata niza dokumentov, ki jih primerjamo, operator plus pa predstavlja konkatenacijo. Z je funkcija za kompresiranje, iz česar lahko razberemo, da vrednost $\mathrm{Z}(a+b)$ dobimo tako, da najprej zlepimo nize obeh dokumentov in kompresiramo rezultat. Z vrača dolžino kompresiranega vhoda v bitih. Manjša vrednost pomeni boljšo kompresijo, ta pa je v splošnem odvisna od tega, koliko besed se v dokumentu ponovi. Več različnih besed pomeni večje število bitov, da zabeležimo stisnjen rezultat. Dokumenti, ki si delijo veliko besed, imajo zato nizko vrednost $\mathrm{Z}(a+b)$, kar pomeni, da so si bolj podobni.

### Jaccardov indeks

Podobnost po Jaccardu je statistika, ki temelji na vrednosti Jaccardovega indeksa [11, 12]. Denimo, da sta dokumenta, ki jih želimo primerjati predstavljena binarno. Ta predstavitev temelji na tem ali dokument vsebuje besedo ali pa je ne. Dokument je tako množica, ki vsebuje samo tiste elemente katerih frekvenca ni enaka nič, oziroma določi nič za vrednost vseh

ostalih elementov. Jaccardov indeks zapišemo kot koeficient števila elementov, ki se pojavijo v obeh dokumentih (moč preseka množic v števcu) in števila dokumentov, ki se pojavijo v vsaj enem od obeh (moč unije množic v imenovalcu):

$$\mathrm{J}(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{2.4}$$

Iz enačbe lahko vidimo, da je vrednost indeksa sorazmerna in raste s številom dokumentov, ki se pojavijo tako v $A$ kot tudi v $B$. Indeks zavzema vrednosti med nič in ena, vrednost ena pa ima tudi v primeru, da operiramo z množicama kjer je moč obeh enaka nič. Ali bomo o podobnosti med dokumenti sklepali na podlagi njihove binarne predstavitve, je odvisno od situacije. Zagotovo, pa je gradnja binarnega modela hitrejša, kot gradnja modela na podlagi frekvenc [13].

### Kosinusna podobnost

Če se zgledujemo po tem, da je kot med vektorjema dober indikator podobnosti, potem je kosinusna podobnost zelo primerna za ugotavljanje podobnosti med dokumenti [14]. Izpeljemo jo lahko iz skalarnega produkta dveh vektorjev:

$$x \cdot y = ||x||\ ||y|| \cos\theta \tag{2.5}$$

Če $a$ in $b$ predstavljata dokumente, ki jih primerjamo, potem mero kosinusne podobnosti lahko zapišemo tako:

$$\text{similarity}(a,b) = \cos\theta = \frac{a \cdot b}{||a||\ ||b||} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}} \tag{2.6}$$

Podobnost določimo iz vrednosti z intervala od -1, ki označuje popolno nasprotje, do 1, ki označuje popolno enakost.

### 2.1.5 Vizualizacija

Kljub avtomatizaciji analize podatkov, ostaja človek pomemben del tega procesa [15]. Računalniki rešijo problem avtomatizacije in shranjevanja ogromnih količin podatkov, za razvoj algoritmov in metod pa je še vedno potrebna človekova ustvarjalnost in sposobnost opazovanja. Najboljši način, na katerega lahko izkoristimo človekovo nadarjenost za prepoznavanje vzorcev, je učinkovita vizualizacija podatkov. Zgovorna vizualizacija lahko veliko pripomore h kvaliteti rezultatov, poleg tega pa za razumevanje pogosto ne zahteva posebne tehnične podkovanosti. Medtem, ko so podatki z veliko šuma za računalnik lahko težavni, jih človek z uporabo dobre vizualizacije lahko identificira brez posebnega truda. Vizualizacije prav tako prispevajo h interaktivnosti vmesnika. Dober uporabniški vmesnik prepoznamo tudi po tem, kako uspešno v tok svojega delovanja vključi uporabnike.

Ko govorimo o prikazu podatkov in rezultatov, imamo v mislih pogosto različne vrste grafov, diagramov in zemljevidov, vendar pa nam za dobro vizualizacijo ni vedno potrebno risati takšnih ali drugačnih slik. Včasih je dovolj, da v podatkih poudarimo ključne elemente. Za primer vzemimo brskalnik, ki je ena izmed preprostejših vendar kljub temu učinkovitih vizualizacijskih tehnik. Kot vhod prejme ključno besedo in nam njene pojavitve v besedilu pobarva, dokumente ki jo vsebujejo pa po možnosti še loči od ostalih. Tako kot pri mnogih drugih pristopih, se analiza besedil tudi pri vizualizacijskih tehnikah zgleduje po podatkovnem rudarjenju. Ko dokumente enkrat prevedemo na podatkovni model, lahko na njem prav tako uspešno uporabimo prikaza z grafom raztrosa ali dendrogramom, kot bi to storili z drugimi strukturiranimi podatki. Dobre vizualizacije koristijo več dimenzij prikaza. S pozicijo v dvodimenzionalnem prostoru, obliko in barvo lahko opišemo že štiri parametre za vsako podatkovno instanco. Zanimiv primer tega lahko vidimo na sliki 2.3.


Slika 2.3: Na sliki vidimo objave Twitter prikazane s spletnim orodjem *Sentiment Viz*. Objave so razporejene po mreži, njihov položaj in barva pa opisujeta sentiment, ki je bil v objavi zaznan. Zeleni krogci desno, označujejo objave z vsebino, ki je naklonjena temi, medtem kot modri krogci levo, označujejo ravno nasprotno. Bolj aktivne objave so narisane višje, kot manj aktivne.

## 2.2 Pregled izbranih orodij

Pridobivanje znanj iz podatkov ni omejeno samo na znanstvena področja. Analize podatkov se morda res lotevamo programsko, vendar njeni ciljni uporabniki še zdaleč niso vsi programerji. S podpora vizualnega programiranja odpravimo potrebo po programerskem znanju, zato je danes v programskih rešitvah za analizo podatkov nepogrešljiva. Velika podjetja se teh rešitev poslužujejo za analizo poslovanja, prepoznavanje trendov in načrtovanje poslovnih taktik. V procesu načrtovanja knjižnice za okolje Orange smo preizkusili nekaj že obstoječih platform za analizo besedil, naša opažanja in uporabniške izkušnje pa so zbrana v tem poglavju.

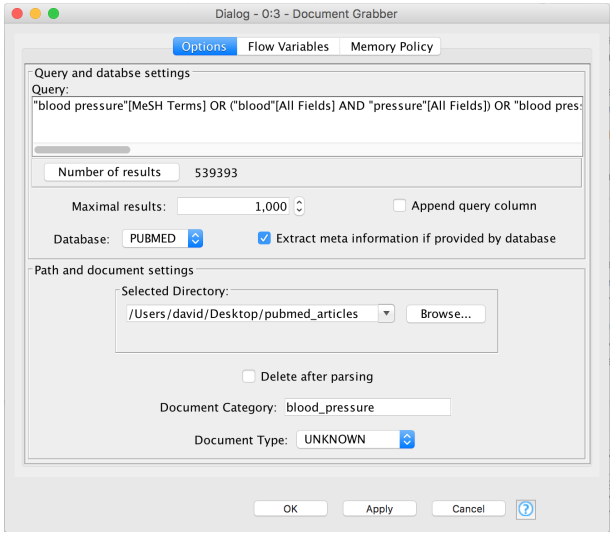
### 2.2.1 Knime

Knime[7] oziroma Konstanz Information Miner je programsko okolje za podatkovno analizo. Zasnovan je bil januarja 2004, razvija pa ga ekipa z Univerze v Konstanzu v Nemčiji. Knime je odprtokodna rešitev in je podprta na vseh večjih operacijskih sistemih. Knime je pravzaprav skupno ime za množico produktov, ki rešujejo podobne probleme[8]. Na primer, Knime Analytics Platform, ki je za nas zanimiv zaradi narave tega magistrskega dela, ima različico imenovano Knime Cloud Analytics Platform, kjer se procesi izvajajo na oblaku. Spoznali smo, da ima podpora izvajanja analitičnih procesov tako lokalno, kot tudi v oblaku v razvoju takih rešitev pomembno vlogo. Pri pregledu orodij, jo bomo omenili kar nekajkrat.

**Gradniki** Tako kot večina večjih orodij, so funkcionalnosti za analizo besedil v Knime na voljo kot dodatna knjižnica. Pri načrtovanju gradnikov za vizualno programiranje v naši knjižnici, smo zato opravili pregled metod, ki jih za to ponuja Knime.

- **Zajem podatkov**
  - Branje *lokalnih* datotek v formatih *.txt*, *.docx*, XML in PDF. Mogoče je tudi branje XML datotek, pridobljenih s spletnega repozitorija PubMed.
  - *Zajem* podatkov z različnih spletnih repozitorijev (v naši različici je bil sicer na voljo samo PubMed).
- **Predobdelava podatkov**
  - *Spreminjanje velikih črk v male* in obratno.
  - *Zamenjava* določenih besed po kriterijih in z besedami navedenimi v slovarju.

[7] `https://www.knime.org/`
[8] `https://www.knime.org/server-products`

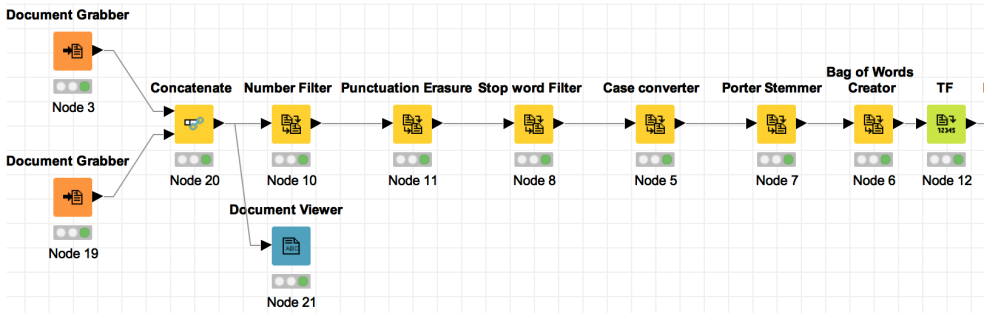



Slika 2.4: Na sliki je prikazan grafični vmesnik Knime gradnika za zajemanje dokumentov s spletnih virov. V tem primeru smo zajeli objave z repozitorija PubMed.

  - *Krnjenje* s Porterjevimi algoritmi (*Porter* in *Snowball*) in algoritmom *Kuhlen*.
  - *Odstranjevanje neinformativnih besed* za več jezikov.
  - *Filtriranje* žetonov na podlagi njihove dolžine, vsebnosti številk ali regularnih izrazov.
  - Gradnja *vreče besed*, z možnostjo uporabe mere TF-IDF.
  - Gradnja predstavitve z $n$-terkami.

- **Vizualizacije**
  - Škatla z brki, histogram, tortni diagram, graf raztrosa, pregledovalnik dokumentov.

**Testiranje platforme** Gradnika *Document Grabber* in *Document Viewer* sta bila za nas še posebej zanimiva. Njuna koncepta sovpadata s koncepti

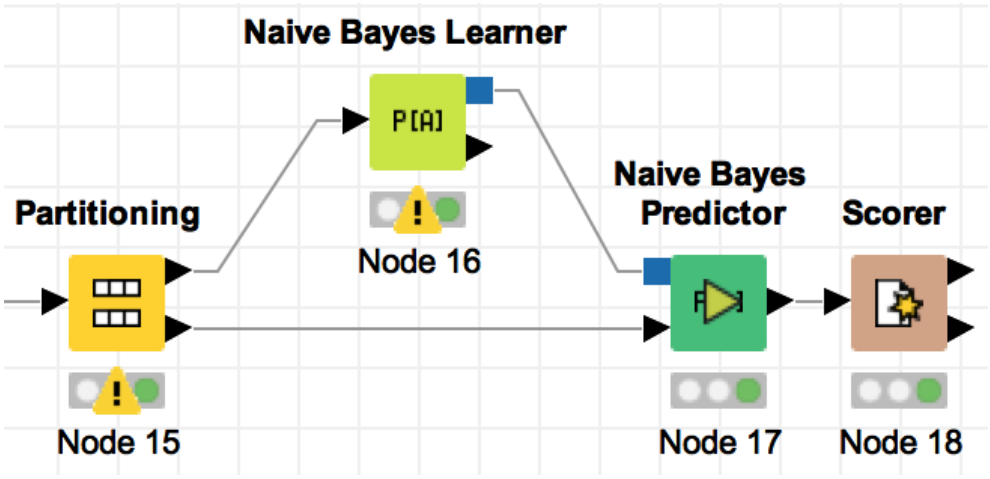



Slika 2.5: Na sliki vidimo proces predobdelave dokumentov v Knime. Večina korakov predobdelave je v Knime definirana z lastnim gradnikom, kar sicer ni slabo, lahko pa se kdaj pojavijo nejasnosti o njihovem vrstnem redu v procesu.

dveh gradnikov, ki smo jih načrtovali implementirati v okolje Orange. Vmesnik gradnik za zajem člankov s PubMeda lahko vidim na sliki 2.4. Vidimo lahko, da podpira iskanje po repozitoriju z uporabo PubMed poizvedovalnega jezika, poleg besedil pa lahko zajamemo tudi metapodatke o objavah. Dokumentom razred določimo ob zajetju (slika 2.4 polje *"Document Category"*). Z uporabo gradnika smo zajeli tisoč člankov na temo krvnega pritiska in jih povezali s pregledovalnikom dokumentov. Pregledovalnik iz zbirke dokumentov ustvari seznam, ki vsebuje metapodatke vseh objav.

Za predobdelavo dokumentov smo ustvarili enostaven proces, ki odstrani številke, ločila in neinformativne besede, prav tako pa žetone krni z algoritmom Porter in na koncu ustvari model vreče besed. Ta proces je v Knime malo daljši kot smo vajeni (slika 2.5), ker moramo za enake rezultate uporabiti več gradnikov. Nekatere gradnike bi morda lahko implementirali kot nastavitve in s tem prihranili prostor v shemi. Nekateri gradniki v Knime omogočajo pripenjanje novi stolpcev v tabelo korpusa. Tako imamo lahko vsak korak predobdelave v lastnem stoplcu, kar omogoča lažjo primerjavo. V našem primeru smo izbrali nastavite, ki uporabi samo en stolpcev in vrednosti sproti prepiše.

Ko z gradnikom *Document vector* ustvarimo vektorske predstavitve dokumentov, lahko podatkovno množico razdelimo na učno in testno (slika 2.6.




Slika 2.6: Na sliki lahko vidimo del procesa v Knime, kjer smo naučili Naivni Bayesov klasifikator. Vmesnik preko obarvanih krogov in ikon uporabniku sporoča stanje gradnikov.

Tu lahko množico razdelimo glede na delež primerov ali pa z navedbo absolutnega števila dokumentov. Razrede smo potem napovedali z uporabo Naivnega Bayesa. Rezultate lahko podamo gradniku *Scorer*, kjer potem prikažemo matriko pravilnih in napačnih uvrstitev.

**Sklep** Knime se morda na prvi pogleda zdi kot zapleteno orodje, vendar ga z malo uporabe hitro osvojimo. Knime uporablja zanimiv pristop k podajanju odzivov o poteku procesa. Pod vsakim gradnikom imamo tri kroge, ki spominjajo na prometne luči. Rdeča označuje, da gradnik ni pripravljen oziroma, da so nastavitve neustrezne. Rumena označuje pripravljenost gradnika za izvajanje, zelena pa nam pove, da se je korak na temu gradniku že izvedel.

### 2.2.2 Rapidminer

RapidMiner[9] je programska rešitev istoimenskega podjetja za gradnjo napovednih modelov in izvajanje statistične analize. Prva iteracija platforme se

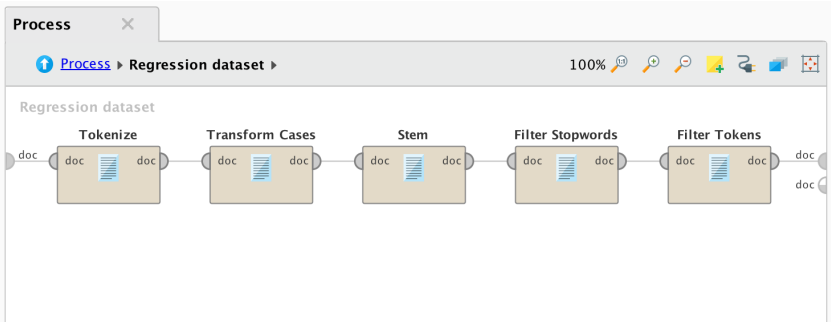
[9] `https://rapidminer.com/`

je imenovala YALE (Yet Another Learning Environment) in se je leta 2006 ob prvem izidu, preimenovala v RapidMiner. Okolje je na voljo v zastonjski in plačljivi različici, je podprto na vseh večjih operacijskih sistemih in nudi tudi možnosti nadgradnje z dodatnimi knjižnicami. Brez dodatkov imamo na voljo približno 400 gradnikov za vizualno programiranje, mi pa smo se pri pregledu osredotočili na knjižnico za analizo besedil.

Ena izmed zanimivih funkcionalnosti je, da ima aplikacija aktivno skupnost uporabnikov, ki jo lahko vključimo v našo uporabniško izkušnjo. Sistemu lahko dovolimo, da nam pri gradnji procesov predlaga gradnike. Predlogi so uteženi glede na popularnost med uporabniki in ustreznost procesu, ki ga gradimo. Platforma dobro poskrbi za nove uporabnike z bogatim izborom testnih projektov, ki so podprti z dokumentacijo in primeri uporabe. RapidMiner razvijalcem ponuja tudi možnost implementacije lastnih knjižnic. Omeniti velja tudi, da okolje omogoča izvajanje procesov tako lokalno kot tudi na oblaku.

**Gradniki** RapidMiner lahko uporabljamo kot vmesnik za vizualno programiranje, ali kot knjižnico v Javi. Mi smo se pri pregledu orodja osredotočili na gradnike za vizualno programiranje. Zanimalo nas je kaj v svoj nabor funkcionalnosti, vključuje eno izmed vodilnih analitičnih platform in kako se bomo po tem zgledovali pri razvoju naše knjižnice.

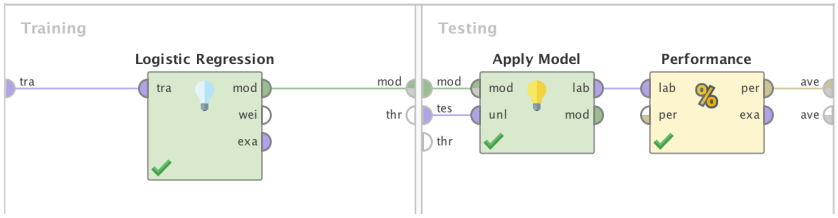
- **Zajem podatkov**
  - Branje iz in pisanje v *lokalne datoteke* v različnih formatih vključno z CSV, XLS, XML, BibtTeX in redkih formatih (angl. *sparse*).
  - Branje in pisanje v *podatkovne baze* SQL.
  - Branje s *spletnih virov* Twitter, Salesforce in predalov za elektronsko pošto preko strežnikov IMAP ali POP3.
  - *Shranjevanje* datotek v oblaku preko storitev Amazon in Dropbox.
  - *Ustvarjanje* in *branje* dokumentov iz nizov, podatkovnih struktur in datotek.




Slika 2.7: Na sliki vidimo proces za predobdelave dokumentov, zgrajenem v okolju RapidMiner. Sestavljajo ga gradnik za razčlenjevanje (angl. *Tokenize*), gradnik za pretvorbo velikih črk (angl. *Transform Cases*), gradnik za krnjenje (angl. *Stem*), gradnik za odstranjevanje neinformativnih besed (angl. *Filter Stopwords*) in gradnik za filtriranje žetonov (angl. *Filter Tokens*).

- **Predobdelava podatkov**
  - *Razčlenjevanje* dokumentov po znakih, besednih vrstah in z regularnimi izrazi. Vključuje tudi razčlenjevanje na stavke.
  - *Prevedba* velikih črk na male.
  - *Odstranjevanje neinformativnih besed* za več jezikov. Vključuje tudi *filtriranje* žetonov glede na njihovo dolžino, vsebino in pripadnost besedni vrsti.
  - *Krnjenje* z algoritmom po Lovinsu in algoritmoma po Porterju (Porter, Snowball). Omogoča tudi definicijo *lastnih pravil za krnjenje*, ki jih podamo v obliki slovarja.
- **Vizualizacije**
  - Graf raztrosa, histogram, stolpični in tortni diagrami, graf porazdelitve.

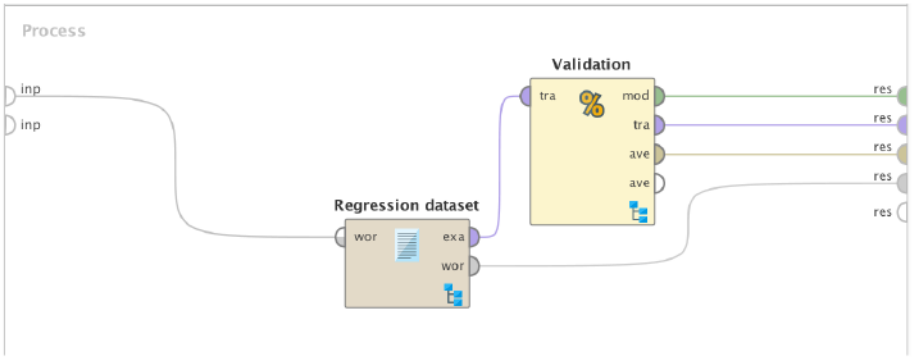
**Testiranje platforme** Orodje smo preizkusili z uvrščanjem člankov New York Times. Članke smo v shemo naložili preko gradnika *Process Documents*




Slika 2.8: V gradniku *Validation* definiramo procesa za učenje (na sliki levo) in testiranje (na sliki desno). Ker smo imeli v našem korpusu samo dva razreda, smo za uvrščanje izbrali logistično regresijo. Rezultate uvrščanja smo ocenili z gradnikom *Performance*.

*from Files*. Gradnik dokumente prebere iz ciljnih map, jim dodeli razred, ki ga določa ime mape in iz korpusov ustvari vektorske modele. Gradnjo modelov opišemo s postopkom predobdelave, ki ga definiramo kot podproces znotraj gradnika (slika 2.7). V *Process Documents from Files* smo naložili mapi *"business_intelligence"* in *"sports"*. Podproces za predobdelavo smo sestavili iz gradnikov za razčlenjevanje, krnjenje, odstranjevanje neinformativnih besed in filtriranje žetonov. Najprej smo vse velike črke pretvorili v male, potem pa smo dokumente razčlenili po znakih, ki niso del abecede (angl. *"non letters"*). Besede smo krnili z algoritmom Lovins in odstranili angleške neinformativne besede. Za konec smo filtrirali še vse žetone krajše od dveh znakov in model zgradili z uporabo mere TF-IDF.

*Process Documents from Files* ima dva izhoda. Na enem vrne podatkovno strukturo *WordList*, ki vsebuje seznam atributov, besed iz katerih so bili ustvarjeni in število pojavitev v dokumentih posameznih razredov. Na drugem vhodu vrne podatkovno strukturo *ExampleSet*, ki vsebuje metapodatke o dokumentih in vektorski model vreče besed. Predobdelane dokumente smo povezali v gradnik *"Validation"* (slika 2.9) v katerem smo korpus ločili na učno in testno množico v razmerju 70 proti 30. Pri tem smo uporabili *stratified sampling*. *Stratified sampling* je vrsta vzročenja, kjer podmnožice zgradimo tako, da so po zgradi podobne primarni množici. Ker uvrščamo samo med dvema razredoma, smo za metodo učenja uporabili logistično regresijo. Za logistično regresijo lahko izbiramo med različnimi optimizacijskimi para-




Slika 2.9: Tu lahko vidimo končni proces za uvrščanje člankov, ki smo ga sestavili v orodju RapidMiner. Hierarhija procesov znatno pripomore k preglednosti celotnega projekta. Podprocese vsebovane v gradnikih *Process Documents from Files* in *Validation* lahko vidimo na slikah 2.7 in 2.8.

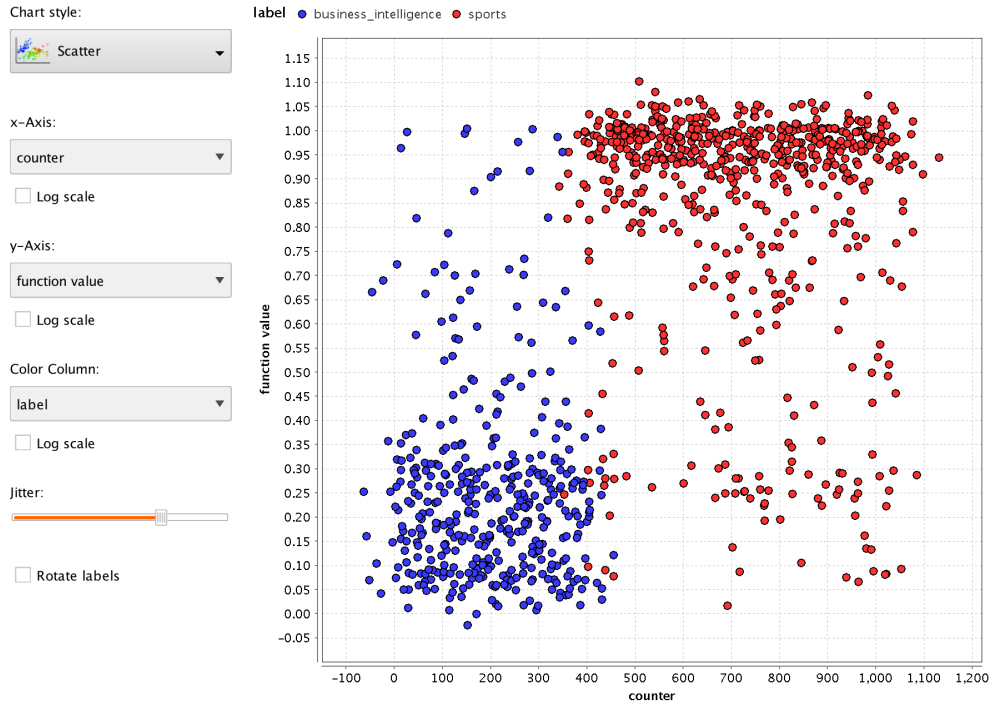
metri. Mejo konvergence (*epsilon*) smo nastavili na $10^{-4}$. Za enačbo jedra smo izbrali opcijo *"dot"*, ki vrednost izračuna po enačbi $\mathrm{k}(x, y) = x * y$. Gradnike za uvrščanje in ocenjevanje smo združili v podproces gradnika *Validation* (slika 2.8).

**classification_error: 24.92%**

| | true business_intelligence | true sports | class precision |
|---|---|---|---|
| pred. business_intelligence | 117 | 75 | 60.94% |
| pred. sports | 3 | 118 | 97.52% |
| class recall | 97.50% | 61.14% | |

Slika 2.10: Tu lahko vidimo tabelo z ocenami, ki jih poda gradnik *Validate*. Uspešnost uvrščanja je opisana z vrednostima točnost (angl. *precision*) in priklic (angl. *recall*).

Iz tabele na sliki 2.10 lahko vidimo, da je bilo uvrščanje športnih člankov uspešnejše, kar smo tudi pričakovali. Zgleda, da vsebina, ki jo opisujejo športni članki, pri uvrščanju vseeno dopušča manj dvoumnosti, kot ta, ki jo vsebujejo članki o poslovanju. RapidMiner nam ob zaključku procesa ponudi tudi nabor vizualizacij, s katerimi lahko prikažemo dobljene rezultate (slika 2.11).




Slika 2.11: Na sliki vidimo rezultate uvrščanja, prikazane z grafom raztrosa (angl. *Scatter plot*). Rdeče pike označujejo članke o športu, modre pa članke o poslovni inteligenci. Na osi $x$ so nanizane zaporedne številke dokumentov v korpusu. Te so pri nekaterih dokumentih zaradi nastavitve *"jitter"* prikazane kot negativne. Na osi $y$ so predstavljene vrednosti funkcije, na podlagi katere smo uvrščali.

**Sklep** V splošnem je RapidMiner zelo zmogljivo orodje. Všeč nam je bil njegov vmesnik in Uspešno poenostavi pristop do tehnologij, ki upravljajo z velikimi podatki, ne da bi pri tem izgubil na zmogljivosti. Na račun tega uživa visoko priljubljenost pri vseh vrstah organizacij, najbol odmeven uporabnik pa je podjetje PayPal. Uporabniki tega ranga namreč želijo po ugodni ceni in s kratkim usposabljanjem pridobiti dobre rezultate, kar pa je ravno to kar RapidMiner ponuja.

### 2.2.3 Textflows

TextFlows[10] je platforma, ki omogoča analizo besedil preko oblačnih storitev. Njen avtor je Matic Perovšek iz oddelka za tehnologije znanja, Instituta Jožefa Stefana. TextFlows je veja projekta ClowdFlows[11], ki je prav tako produkt oddelka za tehnologije znanja in ponuja funkcionalnosti za strojno učenje in podatkovno rudarjenje. Tu najdemo tudi bogat nabor Orange elementov kot so klasifikatorji, več vrst regresije in podporo za uporabo Orange tabel. V naboru funkcionalnosti najdemo tudi nekatere druge produkte inštituta kot je spletna platforma ViperCharts (*Visual Performance Evaluation*)[12] s katero lahko ocenjujemo delovanje klasifikacijskih in napovedovalnih algoritmov. Izstopajoča lastnost vseh teh rešitev je njihova dostopnost. Delujejo namreč v vseh večjih spletnih brskalnikih in za delovanje potrebujejo samo internetno povezavo.

**Gradniki** Pregledali smo nabor gradnikov in funkcionalnosti, ki jih TextFlows ponuja in izpostavili tiste, ki so za nas zanimivi.

- **Zajem podatkov**
  - Vključevanje besedil iz *lokalnih* datotek in *nizov* (v tekstovni in XML obliki).

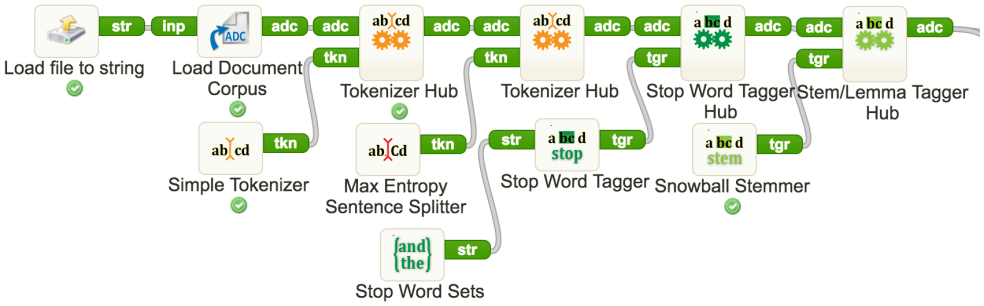
[10] `http://textflows.org/`
[11] `http://clowdflows.org/`
[12] `http://viper.ijs.si/`

  - *Pretvorba* označenih korpusov (angl. *annotated corpus*) v nize in obratno.
  - *Dodajanje* in *urejanje* atributov v korpusih. *Izpis* podatkov o korpusih.
  - *Združevanje* korpusov.

- **Predobdelava podatkov**
  - *Razčlenjevanje* dokumentov po besedah, stavkih ali z regularnimi izrazi. Vključuje tudi širok nabor NLTK implementacij razčlenjevalnikov (med drugimi *Max Entropy*, *Unicode* in *Regex*).
  - *Označevanje besednih vrst* z implementacijami označevalnikov v NLTK, med drugimi *Affix*, *Brill's rule* in *N-gram*.
  - Gradnja modela *vreče besed*.
  - *Krnjenje* s Porterjevimi algoritmi (*Porter* in *Snowball*) in implementacijami algoritmov v NLTK (med drugimi *Regex*, *Lancaster*). Vsebuje tudi NLTK implementacijo lematizacije.
  - *Odstranjevanje neinformativnih besed* za več jezikov.

- **Vizualizacije**
  - Oblak besed, grafi performans ViperCharts, prikazovalnik vsebine korpusov, graf ROC (angl. okrajšava za *Receiver operating characteristic*).

**Testiranje platforme** Pri uporabi platforme smo naleteli na nekaj težav z izvajanjem zgrajenih procesov. Vmesnik v spletnih brskalnikih in izvajanje procesov na strežniku prinese tudi občasne težave z dostopnostjo. Vseeno pa smo lahko sestavili nekaj procesov preizkusili delovanje gradnikov, ki so bili za nas zanimivi. Preden lahko začnemo s predobdelavo dokumentov, moramo v shemo naložiti datoteke v ustreznih formatih. To pomeni, da mora biti vsak dokument predstavljen v svoji vrstici, vrednost pripadajočega razreda



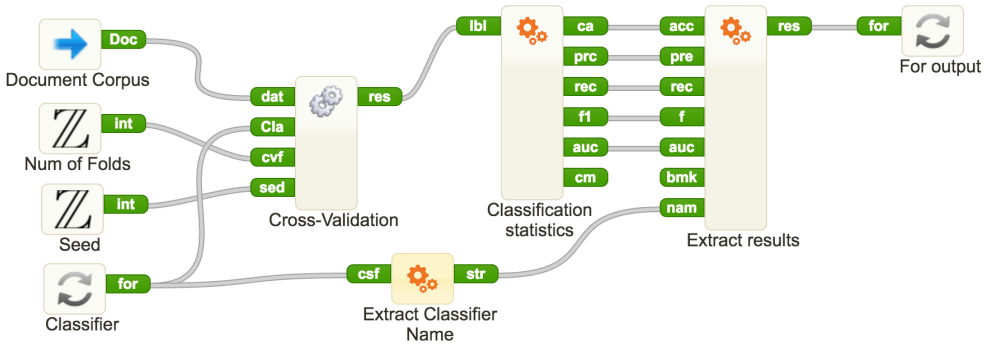

Slika 2.12: Na sliki vidimo proces predobdelave sestavljen v orodju TextFlows. Če primerjamo grajenje shem v orodjih RapidMiner in TextFlows, lahko za slednje rečemo, da je po uporabniški izkušnji nekoliko bolj podobno Orange. V RapidMiner so hierarhije procesov kot elementi toka dela, veliko bolj poudarjene. V večjih primerih to pripomore k preglednosti, za manjše procese pa kdaj učinkuje ravno nasprotno.

pa mora biti napisana kot prva beseda v njej. Pri tem nizu razreda kot predpono, dodamo klicaj (na primer "!sport").

Metode za razčlenjevanje, krnjenje in odstranjevanje neinformativnih v TextFlows uporabljamo preko vmesnih gradnikov (angl. *"hub"*), ki prejmejo dva vhoda (slika 2.12). V TextFlows so vhodi in izhodi označeni z kraticami, ki predstavljajo tip povezave, ki ga sprejmejo. Čeprav je to v RapidMiner rešeno na podoben način, nam je pristop v TextFlows bolj intuitiven. Na vhod "adc" (okrajšava za *annotated document corpus*) podamo korpus, na drugi vhod ("tkn" za *tokenizer* oziroma "tgr" za *stop word/stem tagger*), pa ustrezen gradnik za predobdelavo.

Za ogleda procesa ocenjevanja uvrščanja, smo si ogledali enega izmed dokumentacijski projektov. ta proces je v TextFlows nekoliko bolj zapleten in zahteva uporabo podprocesov. Če želimo primerjati več klasifikatorjev to poteka tako, da gradnike, ki jih predstavljajo združimo v seznam[13], potem pa ta seznam uporabimo kot vhodni podatek v podproces za prečno preverjanje. Ta podproces služi kot zanka *for*, vidimo pa ga lahko na sliki 2.13.

[13]TextFlows vključuje tudi gradnike za gradnjo sestavljenih podatkovnih tipov kot so seznami.




Slika 2.13:

TextFlows podpira tudi upravljanje z osnovnimi podatkovnimi tipi, zato parametre kot je število podmnožic za testiranje (angl. *number of folds*, definiramo v lastnem gradniku in ga povežemo z gradnikom za prečno preverjanje. Podproces nato oceni rezultate uvrščanja, ki jih nato lahko vizualiziramo z grafi kot so ViperCharts.

**Sklep** TextFlows je spodobno orodje za odkrivanje informacij in izvajanje strojnega učenja, vendar v nekaterih primerih njegova prednost lahko postane pomanjkljivost. Ker je delovanje platforme odvisno od delovanja strežnikov in hitrosti internetne povezave, lahko ob neugodnih trenutkih postane nedostopna. Izbiranje gradnikov za shemo se zaradi obsega množice kdaj lahko izkaže za dolgotrajno, vendar širok nabor funkcij, ki so nam na voljo to odtehta. TextFlows zaradi narave svoje arhitekture svojih funkcij ne ponuja v obliki knjižnice, kar pa v končnem pogledu ni pomanjkljivost, saj večino implementacij lahko najdemo v Python knjižnicah kot sta NLTK in Scikit.

### 2.2.4 Diskusija

Z znanjem, ki smo ga pridobili s pregledom področja analize besedil in obstoječih orodij, smo začrtali smernice razvoja naše knjižnice. Identificirali smo štiri glavne komponente, ki služijo kot temelji dobrega sistema za analizo besedil:

1. *Zajem besedilnih podatkov* - Sistem mora imeti dobro definiran in učinkovit način, vpeljave besedil v svoj tok dela. Sem sodi tako vključevanje obstoječih korpusov, kot tudi ustvarjanje novih. Ustvarjanje novih korpusov se nam zdi še celo bolj pomembno, saj to v uporabniku spodbudi iniciativo, kar pa je za orodje kot je Orange, ključnega pomena. Pri tem želimo izpostaviti gradnjo korpusov z dokumenti različnih spletnih repozitorijev. Taki repozitoriji sodijo med dinamične zbirke, kar pomeni, da uporabnik lahko vedno pridobi sveže vnose in s tem ohranja aktualnost vhodnih podatkov.

2. *Predobdelava podatkov* - Zelo pomemben korak v analizi, je predobdelava podatkov. Ta rešuje problem nestrukturiranosti besedil in iz korpusov gradi predstavitvene modele, na katerih lahko izvajamo nadaljne korake. Proces predobdelave mora tako vključevati vsaj razčlenjevanje, spreminjanje velikih črk v male, odstranjevanje neinformativnih besed in krnjenje. Prav tako mora sistem vsebovati vsaj en način, za prevedbo korpusa na vektorsko predstavitev (vreča besed).

3. *Pridobivanje znanj* - Pridobivanje znanj sicer ni končni korak celotnega postopka, lahko pa bi ga v nekem pogledu označili kot njegov cilj. Analize besedil se lotevamo z namenom, da bi iz njih pridobili nova znanja in nove informacije. V tem koraku se lahko zgledujemo po podatkovnem rudarjenju, saj lahko procese za nadzorovano in nenadzorovano učenje, apliciramo tudi na predstavitvene modele, zgrajene iz besedil.

4. *Vizualizacija* - Nepogrešljiv del vsakega sistema, so algoritmi za vizualizacijo podatkov in rezultatov. Kot smo že omenili, je človek kljub avtomatizaciji analitičnih procesov, še vedno velik del analize. Učinkovita vizualizacija je tako odličen način, za vzpostavitev komunikacije med človekovo sposobnostjo prepoznavanja vzorcev in računalnikovo procesorsko močjo.

Kot sklepno točko diskusije, bi izpostavili tudi pomembnost uporabniške izkušnje. Sistem mora biti preprost in intuitiven za uporabo. Vključevati

mora vmesnik za vizualno programiranje, ki bi ga v tem primeru celo postavili pred uporabo njegovega skriptnega dela. Programiranje z gradniki je lahko preprosto in hkrati učinkovito.

# Poglavje 3

# Opis metodologij in uporabljenih razvojnih orodij

## 3.1 Uporabljene tehnologije in knjižnice

V tem poglavju so zbrani opisi tehnologij in orodij, ki smo jih uporabili pri razvoju knjižnice.

### 3.1.1 Python

Python[1] je visokonivojski skriptni programski jezik. Njegov avtor je Guido van Rossum, ki je jezik zasnoval kot naslednik programskemu jeziku ABC. Osnovan je bil proti koncu osemdesetih, prva izdaja pa je bila na voljo februarja 1995. Ideologija jezika povzeta v dokumentu imenovanem *Zen of Python*[2] poudarja berljivost in preprostost programske kode. Enako funkcionalnost lahko v Pythonu dosežemo z manjšim številom vrstic, kot v jezikih Java ali C. Python za gnezdenje ukazov uporablja zamikanje namesto oklepajev, kar pripomore k berljivosti kode, za jezik pa na splošno velja, da se ga lahko hitro naučimo. Podpira več paradigm programiranja vključno z objektnim, imperativnim in funkcijskim.

[1] `https://www.python.org/`
[2] `https://www.python.org/dev/peps/pep-0020/`

Python je dinamično tipiziran jezik. Med implementacijo rešitev je to dobrodošla lastnost, ker nam ni treba skrbeti za ujemanje podatkovnih tipov. Če določena spremenljivka vsebuje niz, njeno vrednost lahko prepišemo s celim številom ali številom v plavajoči vejici. Različne tipe podatkov lahko mešamo tudi v seznamih, slovarjih in ostalih sestavljenih podatkovnih strukturah. Dinamična tipiziranost in sprotno tolmačenje kode sicer vplivata na čas izvajanja kode, vendar je Python v primerjavi z ostalimi skriptnimi jeziki iz te kategorije (Ruby, Javascript) še vedno hitrejši. Zaledje mnogih Google tehnologij je napisano v Pythonu kar dokazuje, da kljub svoji preprostosti ustreza industrijskim standardom [16].

Python ima na voljo bogato standardno knjižnico, na voljo pa je tudi veliko dodatnih modulov in ogrodij. Tiste ki smo jih uproabili za razvoj v podrobneje opišemo v nadaljevanju. Orange 3 je napisan v Pythonu verzije 3, zato smo se tega držali tudi pri razvoju knjižnice. To je zadnja različica Pythona in ni združljiva s prejšnjo (različica 2), se pa danes aktivno uporabljata še obe različici.

### 3.1.2 Javascript

Javascript[3] je skriptni programski jezik, ki se primarno uporablja kot ena izmed temeljnih spletnih tehnologij[17]. Jezik je podprt v vseh večjih brskalnikih, preko katerih posreduje odzive (angl. *input/output*), izvaja pa se na strani klienta (angl. *client-side*). Omogoča več paradigm programiranja, vključno z objektnim, imperativnim in funkcijskim. Razvilo ga je podjetje Netscape, prva različica jezika pa je bil objavljena maja 1995. Pri implementaciji naše knjižnice, smo Javascript uporabili v logiki vizualizacij *Word Cloud* in *GeoMap*.

[3] `https://www.javascript.com/`

### 3.1.3 NLTK

NLTK[4] [18] je Python knjižnica za delo z naravnim jezikom. Zadnja izdaja knjižnice je združljiva s Python verzije 3, še vedno pa je na voljo prejšnja verzija, ki podpira delo z različico 2. Uporaba knjižnice je brezplačna, ker gre za odprtokodni projekt, katerega razvoj poteka s strani uporabniške skupnosti (angl. *community-driven*). V naboru funkcij najdemo metode za predobdelavo in klasifikacijo, vključno s korpusi besedil in množicami manj pomembnih besed za več jezikov. Podrobnosti o tem katere metode smo uporabili v skriptnem delu knjižnice smo navedli v poglavju 4.

### 3.1.4 Gensim

Prve skripte Python knjižnice Gensim[5] [19] so bile najprej del češke knjižnice za digitalno matematiko[6]. Knjižnico je tekom njenega obstoja dopolnjevalo že več razvijalcev, njen originalni avtor pa je Radim Řehůřek. Gensim je odprtokodni projekt in združuje številne funkcije za modeliranje vektorskih prostorov. V modulu lahko najdemo implementacije TF-IDF, algoritmov word2vec in doc2vec, HDP (hierarhični Dirichletovi procesi) in LSA (latentna semantična analiza). Za potrebe našega projekta je bila uporabljena implementacija LDA s katero smo realizirali odkrivanje tem (angl. *topic modeling*).

### 3.1.5 BioPython

Knjižnica BioPython[7] ponuja funkcionalnosti za algoritmično reševanje problemov na področju biologije in bioinformatike. Spisana je v programskem jeziku Python in C, prvotno pa je bila izdana že leta 2000. Vsebuje razrede za shranjevanje, obdelavo in predstavitev biomedicinskih informacij, nas pa

[4] http://www.nltk.org/
[5] https://radimrehurek.com/gensim/
[6] http://dml.cz/
[7] http://biopython.org/

je zanimal vmesnik za dostop do obsežnih podatkovnih baz NCBI (Nacionalni center za biotehnološke informacije). Med te sodijo repozitoriji kot sta GenBank, ki vsebuje sekvence DNK in PubMed, kjer najdemo biomedicinsko literaturo. Za iskanje po teh repozitorijih BioPython vsebuje tudi vmesnik za brskalnik Entrez[8] in razčlenjevalnik za odzive strežnika, ki so v XML ali tekstovni obliki.

### 3.1.6 NumPy

Predhodnik NumPy[9], je bila knjižnica znana pod imenom Numeric. Leta 2005 je Travis Oliphant združil funkcionalnosti knjižnic Numarray in Numeric, ter tako ustvaril NumPy. Gre za odprtokodni dodatek, namenjen delu z visoko dimenzionalnimi polji. NumPy predstavlja velik del Orange implementacije že zaradi Orange tabel. Te so lahko zelo velike, zato je kritično pri tem uporabljati učinkovit modul za manipuliranje z njimi. Funkcionalnosti modula so zgrajene okoli podatkovne strukture *ndarray* (*n-dimensional array*), ki pa lahko vsebuje samo vrednosti enakih podatkovnih tipov, kar za Python ni tipično.

### 3.1.7 PyQt

Qt[10] je odprtokodno ogrodje spisano v C++ in služi razvoju aplikacij in uporabniških vmesnikov za aplikacije. PyQt[11] je verzija tega ogrodja, ki je s povezovalnimi vmesniki prirejena za delo v Pythonu. Razvija ga podjetje Riverbank Computing in produkt ponuja v brezplačni in plačljivi različici. PyQt je podprt na Windows in popularnih Unix sistemih (Linux, OS X) in združuje implementacije osnovnih gradnikov, elementov za povezavo z SQL podatkovnimi bazami, XML razčlenjevalnik in razrede za vpenjanje funkcionalnosti ActiveX za Windows. PyQt platforma podpira delo s Python

[8] `http://www.ncbi.nlm.nih.gov/gquery/`
[9] `http://www.numpy.org/`
[10] `https://www.qt.io/developers/`
[11] `https://riverbankcomputing.com/software/pyqt/intro`

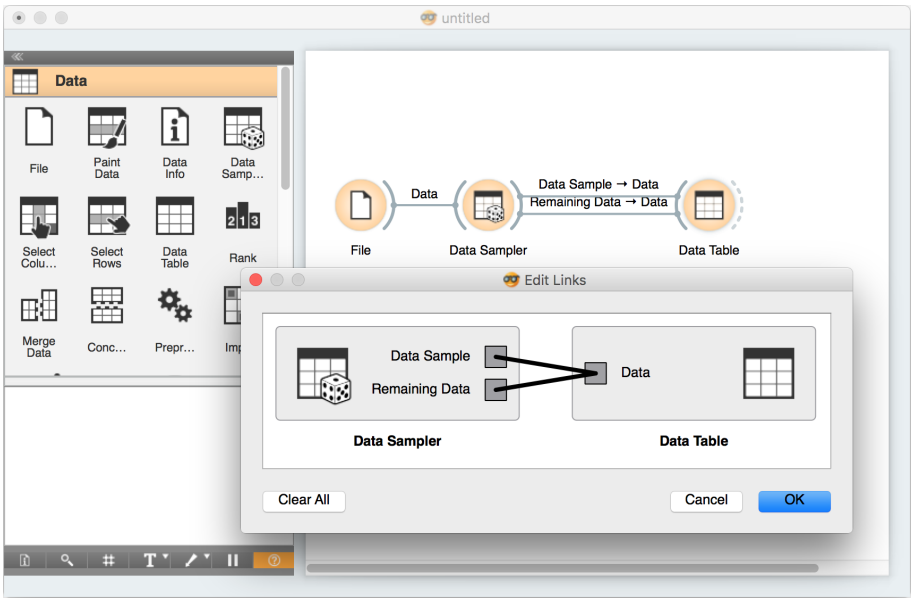
verzije 2 in 3, je redno vzdrževana in dobro dokumentirana. Vmesnik platforme Orange 3 je v celoti zgrajen v PyQt verzije 4.

## 3.2 Orange

Orange[12] je odprtokodna programska platforma za analizo podatkov in izvajanje strojnega učenja [20]. Razvija jo laboratorij za bioinformatiko, Fakultete za računalništvo in informatiko njeni začetki pa posegajo v leto 1997. Orange je bil zasnovan kot C++ knjižnica z algoritmi za strojno učenje in manipulacijo vhodnih podatkov. Izkazalo se je, da pri snovanju aplikacij ni bilo posebne potrebe po uporabi konstruktov C++, zato je jezik implementacije postopoma postal Python. Ta je bil izbran zaradi preprostosti uporabe, fleksibilnosti in kompatibilnosti s kodo v C in C++. Orange tako kot Python v svojem toku dela poudarja učinkovitost in preprostost. Sintaksa v Pythonu je razumljiva in intuitivna, kar začetnikom omogoča hitro osvajanje konceptov, naprednejšim uporabnikom pa omogoča hitro snovanje in razvoj novih konstruktov. Izkazalo se je, da člani skupnosti k platformi raje prispevajo z uporabo Pythona. Prehod na Python je poenostavil tudi razvoj uporabniškega vmesnika, ki je spisan s pomočjo ogrodja PtQt (poglavje 3.1.7).

Večina uporabnikov s platformo upravlja preko grafičnega vmesnika za vizualno programiranje, kar je zgovoren argument o njegovi pomembnosti. V Orange je to ena izmed ključnih funkcij, ki ga v primerjavi z ostalimi Python knjižnicami za strojno učenje postavlja v ospredje. Vmesnik omogoča gradnjo shem z zlaganjem in povezovanje gradnikov preko komunikacijskih kanalov. Kanali so določeni z vhodi oziroma izhodi na vsakem gradniku. Za primer vzemimo gradnik *Data sampler*, ki ga uporabljamo za generiranje podmnožic podatkov iz vhodnih tabel. Z njim lahko vhodne podatke razdelimo na učno in testno množico, vsaka množica pa je potem na voljo kot izhod iz gradnika. Upravljanje s povezavami lahko vidimo na sliki 3.1. Orange gradniki so zaradi

[12] `http://orange.biolab.si/`




Slika 3.1: Nekateri gradniki v Orange imajo na voljo več izhodov. Na sliki vidimo primer, kako oba izhoda gradnika za vzorčenje podatkov povežemo v gradnik za prikaz podatkovne tabele.

preglednosti združeni v razrede, tipičen tok dela pa združuje elemente iz več kategorij.

Standardni nabor metod lahko razširimo z dodatki za Orange. Zaenkrat sta na voljo dodatek za bioinformatiko in dodatek za analizo besedil. Slednjega bomo nadomestili z novo verzijo, ki jo opisujemo v tem magistrskem delu. Za začetne uporabnike, ki z delom v Orange niso seznanjeni so na voljo primeri uporabe in bogata dokumentacija. Za napredne uporabnike, ki želijo večji nadzor nad tokom dela so gradniki so na voljo tudi kot Python razredi. Skriptni del uporabniku omogoča več svobode pri modeliranju toka dela.

# Poglavje 4

# Knjižnica

Orange lahko uporabljamo kot ogrodje za vizualno programiranje ali kot knjižnico v jeziku Python. Naš dodatek za analizo besedil je temu primerno razdeljen na del, ki definira grafične vmesnike gradnikov in del, ki vsebuje njihovo programsko logiko (slika 4.1). Naš doprinos h knjižnici obsega načrtovanje in razvoj gradnikov *Load Corpus*, *New York Times*, *Pubmed*, *Preprocess Text*, *Bag of Words* in *Corpus Viewer*, skupaj z njihovimi pripadajočimi razredi. Razred *Corpus* in gradniki *Word Enrichment*, *Topic Discovery*, *Word cloud* in *Geomap*, skupaj s pripadajočo logiko so delo članov Laboratorija za bioinformatiko.

## 4.1 Korpus v okolju Orange

Tok dela v Orange je osnovan okoli Orange tabele (*Orange.data.Table*). To je podatkovna struktura, ki deluje kot preglednica za podatkovne instance in njihove pripadajoče atribute. Atributi in njihovi tipi so našteti v domeni tabele (*Orange.data.Domain*), njihove vrednosti pa so vsebovane v razredni spremenljivki $X$. $X$ je NumPy matrika dimenzij $n * m$ kjer je $n$ število instanc, $m$ pa število atributov. Vrednosti razredov so vsebovane v razredni spremenljivki $Y$, ki je NumPy vektor dolžine $n$. Na implementaciji Orange tabele smo osnovali razred *Corpus*. Atributi dokumentov so pred predobdelavo

večinoma besedilni, zato so v korpusu shranjeni kot Orange spremenljivke tipa *StringVariable*. Obravnavamo jih kot meta atribute (primer takega korpusa vidimo na sliki 4.2). Ti vsebujejo vrednostih, kot so podatki o avtorjih, datum izida in izvorno besedilo dokumenta. Ko dostopamo do dokumentov korpus objekta, se ti ob klicu metode *documents* zgradijo iz korpusovih meta atributov (slika 4.1).

Izvorna koda 4.1: V tem izseku izvorne kode lahko vidimo metodo razreda *Corpus*, ki iz korpusovih meta atributov zgradi dokumente. To stori s konkatenacijo nizov, ki so vsebovani v meta atributih.

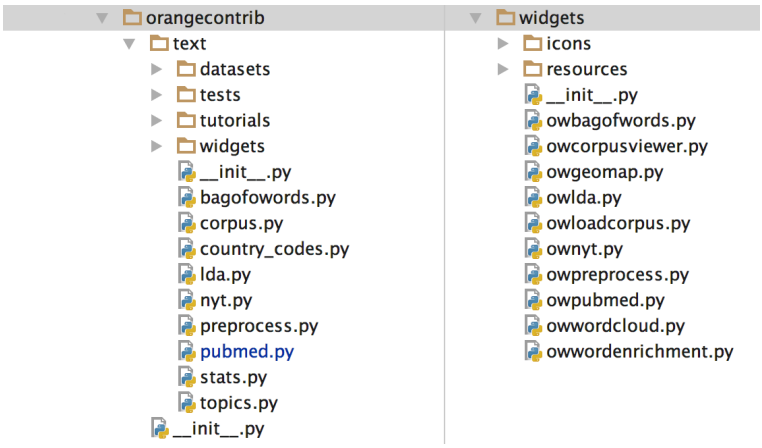
```
def documents_from_features(self, feats):
    data = Table(Domain([], [], [i.name for i in feats],
                        source=self.domain), self)

    text = data.metas.astype(object)
    for col, f in enumerate(data.domain.metas):
        if f.is_discrete:
            for row, val in enumerate(text[:, col]):
                text[row, col] = f.values[int(val)]

    return [' '.join(map(str, i)) for i in text]
```

Matriko $X$ smo v korpusu uporabili za shranjevanje frekvenc žetonov, ki jih pridobimo po generiranju vreče besed. Za lažje delo, smo v razredu *Corpus* definirali tudi nekaj dodatnih metod:

- *documents* - Metoda meta atribute posameznih dokumentov združi v Python seznam nizov. Vsak element seznama predstavlja dokument. Metoda je označena s Python anotacijo *@property*, zato jo lahko pokličemo kot razredni atribut.

- *tokens* - Ta metoda je v delovanju podobna metodi *documents* in vrne seznam razčlenjenih dokumentov. V primeru, da žetoni še niso na voljo, se členitev opravi ob klicu. Ta metoda uporabnikom omogoča, da na vhod gradnikov postavijo neobdelane dokumente, tudi če gradnik za delovanje potrebuje, da so ti razčlenjeni.

- *extend_corpus* - Kadar želimo korpus dopolniti z novimi vnosi, za to uporabimo to metodo. Ta metode se uporablja pri gradnikih *New York*




Slika 4.1: Na sliki vidimo datotečno strukturo knjižnice okolja Orange za analizo besedil, kot jo prikaže razvojno okolje PyCharm. Python datoteke v mapi *text* (na sliki levo) vsebujejo programsko logiko. Definicije uporabniških vmesnikov gradnikov se nahajajo v mapi *widgets* (na sliki desno). Zaradi preglednosti, so datoteke s kodo vmesnikov poimenovane enako kot datoteke s programsko logiko, dodana jim je samo predpona *ow*, ki je okrajšava za *orange widget*.

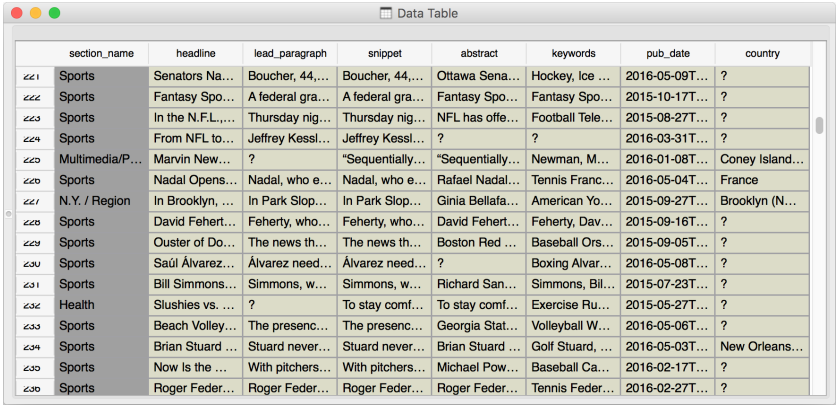
*Times* in *Pubmed*, ki zajemata besedila s spletnih repozitorijev. To poteka postopoma, zato se korpus tekom zajema večkrat dopolni z novimi vnosi.

- *extend_attributes* - Ta metoda deluje podobno kot metoda *extend_corpus*, s to razliko, da namesto instanc, omogoča pripenjanje novih atributov. S spremembo atributov posodobimo tudi domeno korpusa.

- *from_corpus* - Metoda *from_corpus* iz obstoječega korpusa ustvari novega, ki vsebuje dokumente, ki smo jih specificirali z indeksi.

## 4.2 Zajem podatkov

Pri vključevanju vhodnih podatkov v tok dela smo se odločili za dva pristopa: vpeljevanje lokalno shranjenih zbirk in ustvarjanje zbirk z nalaganjem



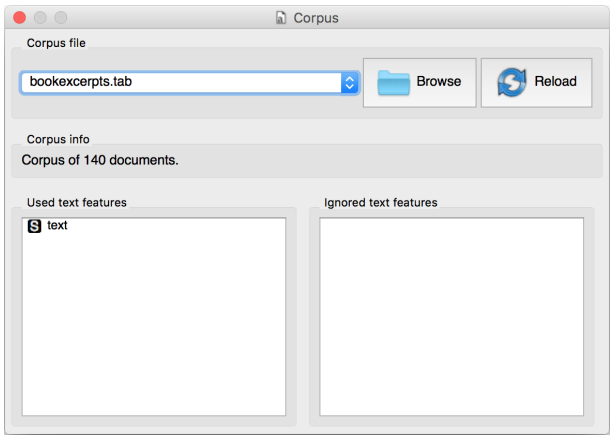

| | section_name | headline | lead_paragraph | snippet | abstract | keywords | pub_date | country |
|---|---|---|---|---|---|---|---|---|
| 221 | Sports | Senators Na… | Boucher, 44,… | Boucher, 44,… | Ottawa Sena… | Hockey, Ice … | 2016-05-09T… | ? |
| 222 | Sports | Fantasy Spo… | A federal gra… | A federal gra… | Fantasy Spo… | Fantasy Spo… | 2015-10-17T… | ? |
| 223 | Sports | In the N.F.L.,… | Thursday nig… | Thursday nig… | NFL has offe… | Football Tele… | 2015-08-27T… | ? |
| 224 | Sports | From NFL to… | Jeffrey Kessl… | Jeffrey Kessl… | ? | ? | 2016-03-31T… | ? |
| 225 | Multimedia/P… | Marvin New… | ? | “Sequentially… | “Sequentially… | Newman, M… | 2016-01-08T… | Coney Island… |
| 226 | Sports | Nadal Opens… | Nadal, who e… | Nadal, who e… | Rafael Nadal… | Tennis Franc… | 2016-05-04T… | France |
| 227 | N.Y. / Region | In Brooklyn, … | In Park Slop… | In Park Slop… | Ginia Bellafa… | American Yo… | 2015-09-27T… | Brooklyn (N… |
| 228 | Sports | David Fehert… | Feherty, who… | Feherty, who… | David Fehert… | Feherty, Dav… | 2015-09-16T… | ? |
| 229 | Sports | Ouster of Do… | The news th… | The news th… | Boston Red … | Baseball Ors… | 2015-09-05T… | ? |
| 230 | Sports | Saúl Álvarez… | Álvarez need… | Álvarez need… | ? | Boxing Alvar… | 2016-05-08T… | ? |
| 231 | Sports | Bill Simmons… | Simmons, w… | Simmons, w… | Richard San… | Simmons, Bil… | 2015-07-23T… | ? |
| 232 | Health | Slushies vs. … | ? | To stay comf… | To stay comf… | Exercise Ru… | 2015-05-27T… | ? |
| 233 | Sports | Beach Volley… | The presenc… | The presenc… | Georgia Stat… | Volleyball W… | 2016-05-06T… | ? |
| 234 | Sports | Brian Stuard … | Stuard never… | Stuard never… | Brian Stuard … | Golf Stuard, … | 2016-05-03T… | New Orleans… |
| 235 | Sports | Now Is the … | With pitchers… | With pitchers… | Michael Pow… | Baseball Ca… | 2016-02-17T… | ? |
| 236 | Sports | Roger Feder… | Roger Feder… | Roger Feder… | Roger Feder… | Tennis Feder… | 2016-02-27T… | ? |

Slika 4.2: Na sliki vidimo korpus odprt v pregledovalniku za tabele. Dokler korpusa ne opišemo z nekim modelom kot je vreča besed, so vrednosti atributov večinoma nizi. V tem primeru je stolpec *section_name* razredni atribut, ostali stolpci pa predstavljajo meta atribute.

s spletnih repozitorijev (New York Times, PubMed). Podpore za prvi primer verjetno ni potrebno posebej utemeljevati. Primerov kjer uporabniki želijo funkcionalnosti platforme uporabiti nad neko lastno zbirko podatkov je verjetno še največ. Za nas bolj zanimiv je drugi pristop, ki omogoča generiranje korpusov iz dokumentov neke dinamične spletne zbirke. Prednosti takega pristopa k zajemu podatkov so sledeče:

- Uporabnik ni omejen na lokalno shranjene podatkovne zbirke.
- Uporabnik se sam odloči katere atribute želi obdržati in tako generira zbirke dokumentov po meri.
- Pridobljeni korpus je takoj pripravljen za delo z Orange, saj za ustrezno strukturo poskrbi logika gradnika.
- Nastavitve zajema se hranijo do naslednje seje, zato uporabnik lahko ustvari enak korpus kot ga je prej.

Na tak način lahko generiramo preprosto zbirko dokumentov za testiranje ali pa obsežen in kompleksen korpus za učenje klasifikatorja. Taki gradniki so zelo fleksibilni, čeprav so odvisni od delujoče internetne povezave. Slednje




Slika 4.3: Prikazno okno gradnika Load corpus.

smo v knjižnici rešili s shranjevanjem rezultatov. Vsi dokumenti, ki smo jih kdaj že ustvarili so shranjene lokalno in so na voljo tudi brez internetnega dostopa. Shranjevanje smo realizirali z uporabo Python modula *shelve*[1].

### 4.2.1 Load Corpus

Če uporabljamo skriptni del knjižnice, lahko korpuse definiramo neposredno z uporabo Orange razreda *Corpus*. V primeru, da uporabljamo vmesnik za vizualno programiranje, pa korpus v shemo lahko vpeljemo z uporabo gradnika *Load Corpus*. Datoteke korpusov izberemo s pomočjo vgrajenega brskalnika. Ko izberemo ciljno datoteko se izbor shrani v zgodovino uporabnikovih izbir, ki je dostopna preko spustnega menija. Ker Orange korpus osnovan na implementaciji Orange tabele, gradnik na vhodu sprejme datoteke s pripono *.tab*, kar označuje datoteke katerih vsebina je ločena z zamiki (angl. *tab delimited file*). V primeru, da se je naložena datoteka tekom dela spremenila, lahko njeno vsebino posodobimo s klikom na gumb "Reload". Če nimamo na voljo nobenih ustreznih datotek, lahko preko brskalnika izberemo katerega izmed dokumentacijskih korpusov.

Ko se izbrana datoteka naloži, lahko izberemo atribute, s katerimi želimo

[1] `https://docs.python.org/2/library/shelve.html`

delati. Levi seznam vsebuje atribute, ki jih želimo v korpusu obdržati, v desnega pa lahko premaknemo atribute, ki jih želimo izpustiti. Seznama podpirata koncept "drag and drop" zato lahko atribute med množicama premikamo z enim samim gibom miške. Gradnik vhodov nima, ima pa en sam izhod na katerega posreduje naloženi korpus.

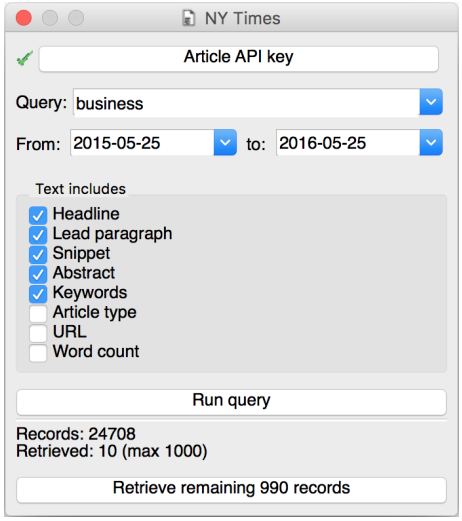
### 4.2.2 New York Times

Razred *NYT* je namenjen iskanju in zajemu besedil z novičarskega portala New York Times[2]. Portal New York Times razvijalcem ponuja API vmesnike (angl. *application programming interface*) za pridobivanje različnih vsebin. Za našo knjižnico smo se odločili uporabiti vmesnik za iskanje objavljenih člankov. Članki so odraz dogajanj v svetu in so zato tudi dobro izhodišče za iskanje in odkrivanje trendov.

Izvorna koda 4.2: Ta izsek iz izvorne kode demonstrira pošiljanje zahtev na New York Times API točke. Ker parametre klica podajamo v URL naslovu jih moramo kodirati v ustrezen format. Ko se strežnik odzove z JSON slovarjem, lahko njegovo vsebino ustrezno razčlenimo.

```
import json
from urllib import request, parse

base_url = 'http://api.nytimes.com/svc/search/v2/
    articlesearch.json'
parameters = parse.urlencode([
    ('q', 'sports'),
    ('fq', 'The New York Times'),
    ('api-key', 'key'),
    ('fl', 'headline')
])
query_url = '{}?{}'.format(base_url, parameters)

with request.urlopen(query_url) as connection:
    response = connection.read().decode('utf-8')
    response_object = json.loads(response).get('response')
    for document in response_object.get('docs'):
        # Branje dokumenta ...
```

[2] http://www.nytimes.com/




Slika 4.4: Na sliki vidimo vmesnik za upravljanje z New York Times gradnikom. Parametre iskanja določimo z vnosom vrednosti oziroma izbiro ustreznih polj. Polja podpirajo beleženje uporabnikovih prejšnjih vnosov.

New York Times API vmesnik za uporabo zahteva API ključ. API ključi se uporabljajo za identifikacijo pošiljatelja zahteve. Ker ima Orange veliko uporabnikov, ne bi bilo primerno, da bi vsi uporabljali enak ključ. To smo rešili tako, da mora uporabnik za uporabo gradnika in skriptnega dela, vnesti lasten (in veljaven) New York Times API ključ. API točka za poizvedbe po člankih se odziva na zahteve vrste GET. To pomeni, da moramo za vsako zahtevo sestaviti ustrezen URL, ki s parametri definira poizvedbo (primer kode 4.2). Za pošiljanje zahtev smo uporabili Python modul urllib[3].

Poizvedbe v Orange *NYT* opišemo s ključnimi besedami in želenim časovnim intervalom. V gradniku lahko preko potrditvenih polj, izbiramo katere dele člankov želimo prenesti[4] (slika 4.4, nastavitve pod oznako *Text includes*). Izbrana polja so potem v generiranem korpusu na voljo kot meta atributi v Orange domeni. New York Times API vmesnik za pridobivanje člankov, se na zahteve odziva s podatki v formatu JSON (primer kode 4.2 za primer branja JSON odziva). Člankom se ob zajemu definira razred, ki ga določimo

[3] `https://docs.python.org/2/library/urllib.html`
[4] Če uporabljamo metode v skriptnem delu, se prenesejo vsi.

na osnovi vrednosti polja "section name", če je ta na voljo. V nasprotnem primeru, se smatra, da ti članki nimajo razreda (angl. *missing values*).

Zajem podatkov preko gradnika poteka v dveh korakih. V prvem pridobimo samo informacije o člankih, v drugem pa te članke zajamemo. Če uporabimo metodo objekta *NYT*, lahko zajem opravimo v enem samem koraku (primer kode 4.3). Pri uporabi New York Times API vmesnika, smo morali upoštevati omejitve, ki jih predpisuje. Kot uporabniki ne smemo poslati več kot pet zahtev na sekundo, omejeni pa smo na tisoč klicev na dan. Članki, ki jih vrne posamezna poizvedba so razporejeni po več URL naslovih. Na posameznem naslovu se nahaja največ deset člankov, kar pomeni, da moramo za zajem tisoč člankov opraviti sto klicev. Med zaporedne klice smo zato dodali kratek premor, ker bi se ti sicer izvajali prehitro. Zaradi vmesnih premorov lahko zajem traja kar nekaj časa, zato smo uporabniku omogočili, da zajem ustavi s pritiskom na gumb. Na ta način ostanemo znotraj omejitev API vmesnika, uporabnik pa ima nadzor nad tem kdaj želi prenehati s prenosom člankov.

Izvorna koda 4.3: Na tem primeru lahko vidimo, kako s svojim API ključem ustvarimo NYT objekt in tako pridobimo Orange korpus, ki vsebuje iskane članke.

```
from datetime import date
from orangecontrib.text.nyt import NYT

api_key = 'apikey'
nyt_object = NYT(api_key)

nyt_corpus = nyt_object.run_query(
    query='politics',
    date_from=date(2016, 5, 1),
    date_to=date(2016, 6, 1),
    max_records=100  # Koliko dokumentov zelimo prenesti.
)

domain = nyt_corpus.domain
documents = nyt_corpus_documents
# Nadaljujemo delo z Orange korpusom ...
```

New York Times gradnik iz prenešenih člankov ustvari Orange korpus, ki je

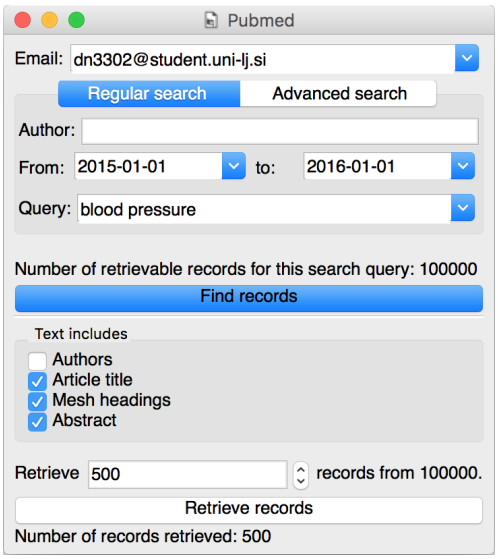
na voljo kot izhod iz gradnika. New York Times članke lahko pridobimo tudi s skriptnim delom knjižnice (primer kode 4.3 za primer uporabe).

### 4.2.3 Pubmed

Orange razred *Pubmed* je namenjen iskanju in zajemu podatkov z istoimenskega, spletnega repozitorija za biomedicinsko literaturo. Za dostop do repozitorija smo uporabili vmesnik Python knjižnice Biopython. Ta za dostop do PubMed repozitorija, zahteva samo identifikacijo z naslovom za elektronsko pošto.

Izvorna koda 4.4: Na tem primeru izvorne kode, vidimo zajem podatkov z Biopython vmesnikom za Entrez in z Orange razredom *Pubmed*. Korake zajema smo v *Pubmed* združili v en klic in s tem poenostavili uporabo. Parametri metod so ponekod zaradi preglednosti izpuščeni.

```
from Bio import Entrez, Medline
from orangecontrib.text.pubmed import Pubmed

# 1. Postopek zajema podatkov z Biopython.
email = 'user_email'
Entrez.email = email

search_results = Entrez.read(Entrez.esearch(...))
post_results = Entrez.read(Entrez.epost(...))
web_env = post_results.get('WebEnv')
key = post_results.get('QueryKey')
data = Medline.parse(Entrez.efetch(webenv=web_env, query_key=
    key, ...))
for record in data:
    # Preberemo dokumente ...
fetch_handle.close()

# 2. Zajem podatkov v Orange.
pubmed_object = Pubmed(email)
pubmed_corpus = pubmed_object.download_records(
    terms=['blood pressure'],
    pub_date_start='2015/01/01',
    pub_date_end='2016/01/01',
    num_records=100
)
for document in pubmed_corpus.documents:
    # Preberemo dokumente ...
```




Slika 4.5: Grafični vmesnik gradnika *Pubmed* omogoča sestavljanje poizvedb preko vnosnih polj, zahteve pa lahko opišemo tudi s sintakso PubMed poizvedovalnega jezika. Potrditvena polja pod oznako "Text includes", določajo katere dele zapisov želimo imeti v korpusu kot atribute.

Da zahteve ne bi bile prepogoste, je zaželjeno, da poizvedbe izvajamo v dveh korakih[5]. V prvem pridobimo podatke o zapisih, v drugem pa opravimo prenos podatkov. Tako ima strežnik dovolj časa, da pripravi rezultate. V skriptnem delu naše knjižnice smo zaradi preprostosti vse te korake združili v en klic (primer kode 4.4). Na spletni strani NCBI[6] (National Center for Biotechnology Information) lahko po vsebini baze PubMed iščemo s posebnim poizvedovalnim jezikom[7]. V gradnik razreda *Pubmed* smo zato vključili dva načina iskanja zapisov: preko vnosnih polj in spustnih menijev, ali z zahtevo, ki jo sestavimo v omenjenem poizvedovalnem jeziku (slika 4.5, zavihka "Regular/Advanced search"). Opisovanje poizvedb je podobno kot pri razredu *NYT*, s to razliko, da tu lahko iščemo tudi po avtorjih besedil. Biopython vmesnik za Entrez zapise vrača v XML oziroma tekstovni obliki. Oba for-

[5] `http://www.ncbi.nlm.nih.gov/books/NBK25497/`
[6] `http://www.ncbi.nlm.nih.gov/pubmed`
[7] `http://www.ncbi.nlm.nih.gov/pubmed/advanced`

mata lahko preberemo z razčlenjevalnikom Biopython modula Medline. Pri ustvarjanju Orange korpusa, logika dokumentom določi razred na podlagi vrednosti polja "MeSH" (angl. *Medical Subject Headings*), če je ta prisoten v zajetih zapisih. Če ni, se privzame, da ti dokumenti ne pripadajo nobenemu razredu.

## 4.3 Predobdelava podatkov

Za predobdelavo podatkov smo v knjižnico vključili razreda *Preprocess* in *BagOfWords*. *Preprocess* vsebuje metode za predobdelavo besedil, *BagOfWords* pa uporabljamo za gradnjo modela vreče besed.

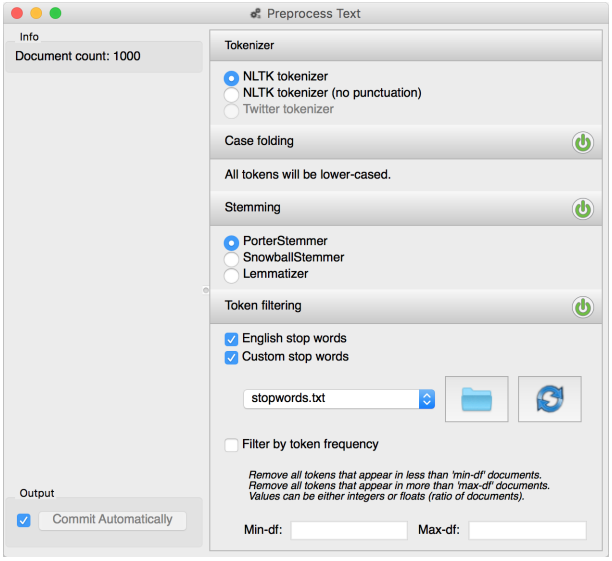
### 4.3.1 Preprocess

*Preprocess* je razred za predobdelavo dokumentov v Orange korpusih. Zasnovan je kot zaporedje funkcij, kjer se na vsakem koraku nad besedilom izvede en del obdelave. Če želimo, lahko posamezne korake (z izjemo razčlenjevanja[8]) iz postopka izpustimo. Vmesnik gradnika smo zato opremili s stikali, preko katerih lahko posamezne korake predobdelave omogočimo oziroma onemogočimo (slika 4.6). Proces predobdelave smo ločili na štiri (zaporedne) korake: razčlenjevanje, spreminjanje velikih črk v male, krnjenje oziroma lematizacija in filtriranje neinformativnih atributov.

#### Razčlenjevanje

NLTK ponuja več vrst metod za razčlenjevanje besedil. Bolj preprosti, členijo samo po presledkih, nekoliko naprednejše, za to uporabljajo regularne izraze. NLTK nudi tudi razčlenjevalnike prirejene specifičnim vrstam besedil (recimo Twitter objave). Prvi razčlenjevalnik ki smo ga vključili je *Penn Treebank*

[8]Brez razčlenjevanja besedil bi bili nadaljnji postopki predobdelave nesmisleni, prav tako pa ne bi mogli graditi predstavitvenih modelov.




Slika 4.6: Vmesnik gradnika *Preprocess Text*.

razčlenjevalnik[9]. Izraz Treebank označuje podatkovno bazo v kateri hranimo drevesne strukture besedil. Te so zgrajene okoli sintaktičnih in semantičnih značilnosti besednih vrst. Z uporabe te baze in regularnih izrazov lahko ta razčlenjevalnik zazna tudi okrajšave besednih zvez kot sta *"can't"* (zdaj *"ca"* in *"n't"*) in *"don't"* (zdaj *"do"* in *"n't"*).

Ker *Penn Treebank* razčlenjevalnik kot veljavne žetone vrača tudi ločila, smo dodali še alternativno možnost, ki ločila filtrira. V ta namen smo uporabili NLTK *RegexpTokenizer* razčlenjevalnik. Za uporabo tega razčlenjevalnika, moramo podati lasten regularni izraz. V našem primeru smo definirali takega, ki filtrira vse, kar ni eden ali več zaporednih abecednih znakov (primer kode 4.5).

Izvorna koda 4.5: Uporaba NLTK RegexpTokenizer razčlenjevalnika s prirejenim regularnim izrazom.

```
1 from nltk.tokenize import RegexpTokenizer
2
3 tokenizer = RegexpTokenizer(r'\w+')
```

[9]Penn Treebank je prva obsežna podatkovna baza dreves besedilnih struktur. Razvila jo je univerza v Pennsylvaniji.

### Krnjenje in lematizacija

Za transformacijo besed smo v razred *Preprocess* vključili dva algoritma za krnjenje in enega za lematizacijo. Uporaba algoritmov je zelo odvisna od podatkov in procesa v katerega jih želimo vključiti[10], zato smo se odločili vključiti oba. Za krnjenje lahko izbiramo med NLTK implementacijami obeh Porterjevih algoritmov *PorterStemmer* in *SnowballStemmer*). Slednji je izboljšana različica prvega in pri krnjenju tipično odstrani večji del besede[11]. Uporabljamo ga lahko za več jezikov, lahko pa tudi izberemo nastavitev naj neinformativnih besed ne krni

## 4.3.2 Filtriranje atributov

Po razčlenjevanju in krnjenju, smo v proces dodali še možnost odstranjevanja neinformativnih besed. V razredu *Preprocess* lahko vir neinformativnih besed definiramo s tremi različnimi vrednostmi.

Izvorna koda 4.6: V izseku kode vidimo kako z uporabo razreda *Preprocess* predobdelamo dokumente Orange korpusa. Izbrali smo privzeti razčlenjevalnik, krnjenje z metodo Porter in bazo angleških neinformativnih besed. Odstranili smo tudi žetone, ki se pojavijo v več kot 30% in manj kot 10% dokumentov.

```
from orangecontrib.text.corpus import Corpus
from orangecontrib.text.preprocess import (Preprocessor,
                                           PorterStemmer)

corpus = Corpus.from_file('bookexcerpts')
pp_object = Preprocessor(
        tokenizer='default'
        stop_words='english',
        min_df=0.1, max_df=0.3,
        transformation=PorterStemmer
)
pp_corpus = pp_object(corpus)
print(pp_corpus.tokens[0])  # Krnjeni zetoni prvega dokumenta.
# ['hous', 'jim', 'say', ...]
```

[10]Lematizacija je časovno potratnejša kot krnjenje.

[11]http://www.nltk.org/howto/stem.html

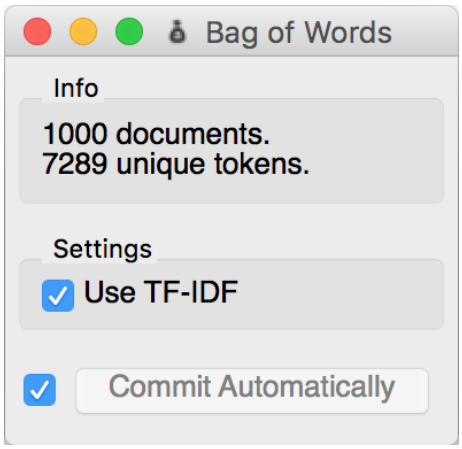
Če besed ne želimo odstraniti, potem vrednost parametra pustimo prazno. Če želimo odstraniti besede za katerega izmed jezikov, ki jih podpira NLTK, potem podamo ime jezika[12]. Definiramo lahko tudi lastno množico neinformativnih besed. Če uporabljamo skriptni del, jo podamo kot Python seznam, če pa uporabljamo gradnik za predobdelavo, je ta zato opremljen z datotečnim brskalnikom (slika 4.6). Datoteke morajo biti v formatu *.txt*, vsaka beseda pa mora biti zapisana v svoji vrstici. Besede lahko odstranimo tudi na podlagi najmanjše in največje pojavitve v dokumentih. Pri tem vrednosti *max_df* in *min_df* določimo procentualno (z vrednostmi v plavajoči vejici), ali pa s celimi števili. Za uporabo mere je dovolj, da definiramo eno izmed mej.

### 4.3.3 Bag of Words

Za generiranje predstavitvenih modelov besedil, smo v dodatek vključili razred *BagOfWords*, ki iz Orange korpusa ustvari vrečo besed. $BagOfWords$ je povezovalni člen med objektom razreda *Corpus* in ostalimi funkcionalnostimi v Orange. Njegova naloga je dokumente korpusa prevesti na tako obliko, da nad njimi lahko izvedemo postopke strojnega učenja in klasifikacije. Ker predstavitvenih modelov ne moremo graditi pred členitvijo dokumentov, smo v gradnik vključili osnovno predobdelavo. Ta se izvede v primeru, da dokumenti korpusa še niso bili razčlenjeni. V tem primeru, se nad vhodnim korpusom izvede samo razčlenjevanje s privzetim razčlenjevalnikom, rezultati pa so zgotovo boljši, če podatke predobdelamo z uporabo naprednejših nastavitev razreda *Preprocess*.

Ker večino predobdelave opravimo prav z razredom *Preprocess*, *BagOfWords* ne potrebuje veliko parametrov. Zato smo se odločili za minimalističen vmesnik (slika 4.7). Med nastavitvami smo vključili samo možnost za uporabo mere TF-IDF. Modeli ustvarjeni s TF-IDF so bistveno bolj informativni, kot taki, ki temeljijo samo na številu pojavitev besed.

[12]NLTK vsebuje baze neinformativnih besed za 11 jezikov[18].




Slika 4.7: Vmesnik gradnika *Bag of Words*.

Model vreče besed smo generirali z uporabo knjižnice Gensim. Najprej smo uporabili Gensim razred *Dictionary*, ki iz korpusovega seznama žetonov ustvari besedišče (angl. *vocabulary*). Žetone predstavi z zaporednimi številskimi vrednostmi in prešteje v koliko dokumentih se pojavijo.

Izvorna koda 4.7: Ta izsek iz programske kode opisuje gradnjo modela vreče besed iz Orange korpusa.

```
from gensim import corpora
from orangecontrib.text.preprocess import (Preprocessor,
                                           PorterStemmer)

test_corpus = Corpus.from_file('deerwester')
preprocessor = Preprocessor(
    tokenizer='no_punct',
    stop_words='english',
    transformation=PorterStemmer
)
pp_corpus = preprocessor(test_corpus)

tokens = pp_corpus.tokens
dictionary = corpora.Dictionary(tokens)
bag_of_words = [
    dictionary.doc2bow(document) for document in tokens
]
print(bag_of_words[0])
# [(0, 1), (1, 1), (2, 1), (3, 1), ...]
```

Model vreče besed smo nato generirali z metodo *doc2bow*. Ta vsak dokument predstavi s seznamom parov številskih vrednosti. Prva vrednost v paru je žeton predstavljen z zaporedno številko, ki mu pripada v besedišču, druga vrednost pa je število njegovih pojavitev v dokumentu. Postopek lahko vidimo na primeru programske kode 4.7.

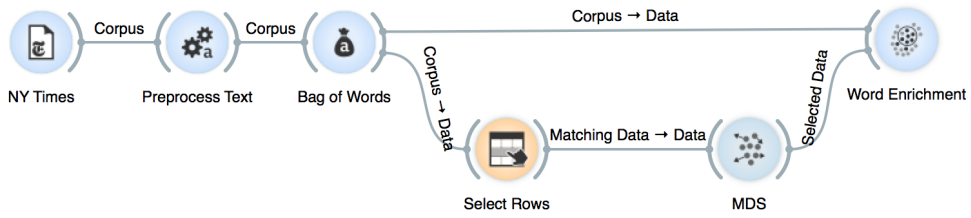
Z uporabo metode *corpus2dense* razred nato ustvari matrično predstavitev korpusa in jo zapiše v spremenljivko $X$ korpus objekta. V matriki so vrednosti shranjene v formatu *numpy.float64* (plavajoča vejica z dvojno natančnostjo). Na ta način jih lahko v primeru uporabe TF-IDF, zaokrožimo na ustrezno število decimalnih mest. Za gradnjo modela TF-IDF smo uporabili implementacijo Gensim razreda *TfidfModel*.

## 4.4 Analiza

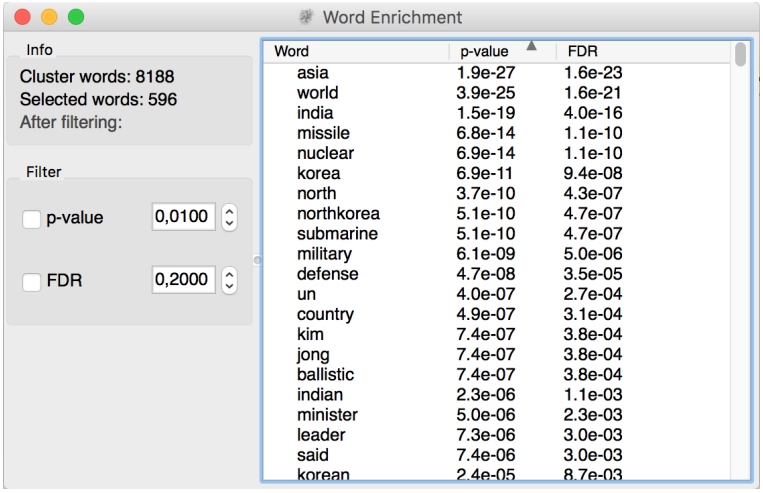
Med analitične widgete naše knjižnice smo vključili gradnik za odkrivanje karakterističnih besed (*Word Enrichment*) in gradnik za odkrivanje tem v besedilih (*Topic Modeling*).

### 4.4.1 Word Enrichment

Delovanje *Word Enrichment* lahko razumemo kot iskanje karakterističnih besed. Recimo, da bi radi izvedeli, kaj so značilne besede za članke o tehnologiji. Iz korpusa člankov najprej zgradimo vrečo besed, nato pa izberemo tiste dokumente, ki pripadajo razredu *"technology"*.




Slika 4.8: Slika prikazuje primer sheme, kjer z *Word Enrichment* iščemo značilne besede za članke zajete z *New York Times*.




Slika 4.9: Slika prikazuje rezulate iskanje karakterističnih besed z *Word Enrichment*. Med rezultati vidimo značilne besede za New York Times članke na temo *"world"*. Izbrali smo jih v gradniku *MDS* na podlagi njihove umestitve v grafu.

Tu bi lahko uporabili tudi gradnik *MDS* za večrazsežnostno lestvičenje (angl. *multidimensional scaling*). Z njim lahko vizualiziramo dokumente in izberemo skupino relativno podobnih člankov (na grafu poiščemo gruče točk, ki se držijo skupaj). Izbrane članke in celoten korpus potem lahko povežemo z gradnikom za bogatenje besed (slika 4.8).

Določanje karakterističnih besed poteka z izračunom funkcije $p$ in mero imenovano *False Discovery Rate* (okrajšano kot FDR). Vrednost $p$ izračunamo za vsako besedo in s tem dobimo vrednost, ki nam poda informacijo o "skrajnosti" te besede, kot nekega opazovanega izida. Z drugimi besedami, $p$ je verjetnost, da bo pri opazovanju ta izid vsaj enak, ali pa še bolj skrajen. Bolj karakteristične besede imajo manjše vrednosti $p$. Za vsako besedo, gradnik izračuna tudi vrednost FDR (slika 4.9). Ta nam poda pričakovani odstotek napačnih izidov za to besedo. Rezultate lahko na podlagi vrednosti $p$ in FDR tudi filtriramo.

### 4.4.2 Topic Modeling

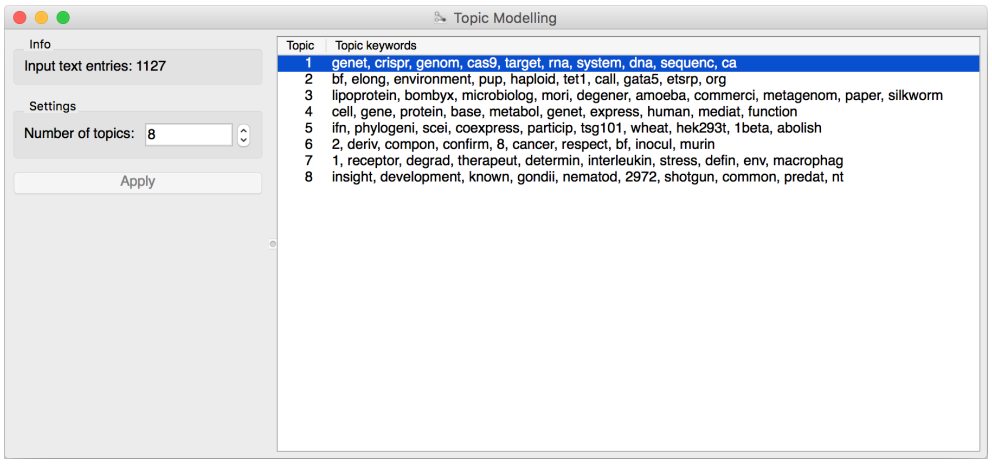
Orange razred *LDA* je namenjen odkrivanju tem z latentno Dirichetovo alokacijo. Latentna Dirichletova alokacija (okrajšano LDA) je generativen statistični model, ki dokumente predstavi kot množico tem [21]. Množice imajo za vsako vsebovano temo, določeno neko verjetnost, s katero bodo generirale besede, ki pripadajo tem temam. Algoritem z upoštevanjem te logike, na podlagi dokumentov poskuša najti nabor tem, za katere je najbolj verjetno, da so ustvarile tako zbirko.

Pri snovanju gradnika smo uporabili implementacijo algoritma iz knjižnice Gensim. Objekt razreda *LDA* pokličemo s seznamom žetonov ciljnega korpusa in številom tem, ki jih želimo odkriti. Gradnja modela za odkrivanje, se začne z gradnjo vreče besed iz vhodnega korpusa. Tu gre za enak proces, ki ga uporablja tudi razred $BagOfWords$ (primer kode 4.7). Ob klicu razred vrečo besed generira sam, kar pomeni, da uporabnik na vhod lahko poda kar Orange korpus brez predobdelave. Matrična predstavitev korpusa, se potem uporabi kot vhodni podatek v Gensim razred $LdaModel$. Eden izmed opcijskih parametrov je število obhodov čez korpus.

Izvorna koda 4.8: Na tem primeru lahko vidimo gradnjo LDA modela z uporabo Gensim razreda $LdaModel$. Spremenljivka *tokens* je seznam seznamov, ki vsebuje žetone dokumentov. Metoda *chunk_list* vhodni seznam razdeli na $n$ podseznamov dolžine 100, s katerimi potem gradimo in posodabljamo model.

```
from gensim import corpora, models

tokens = [ ... ]  # Seznam seznamov, ki vsebujejo zetone.
dictionary = corpora.Dictionary(tokens)
corpus = [dictionary.doc2bow(token) for token in tokens]

lda_model = models.LdaModel(id2word=dictionary, num_topics=3)
for i, part in enumerate(chunk_list(corpus, 100)):
    lda_model.update(part)  # Posodobimo model.
```

Večje število obhodov tipično vrne boljše podatke, vendar je za večje korpuse časovno potratno. V implementaciji razreda *LDA*, smo uporabili privzeto




Slika 4.10: Prikaz tem v Orange gradniku *Topic Discovery* za korpus PubMed člankov na temo encima Cas9.

vrednost, kar je 50 obhodov. Za zahtevnejše naloge, je v Gensim na voljo tudi paralelna implementacija algoritma[13]. Model za odkrivanje se gradi postopoma. Logika iterira čez skupine dokumentov in s klicem metode *update* posodablja model (primer kode 4.8). Gradnik ima dve izhodni povezavi. Na eni se nahaja vhodni korpus, na drugi pa seznam tem s pripadajočimi utežmi.

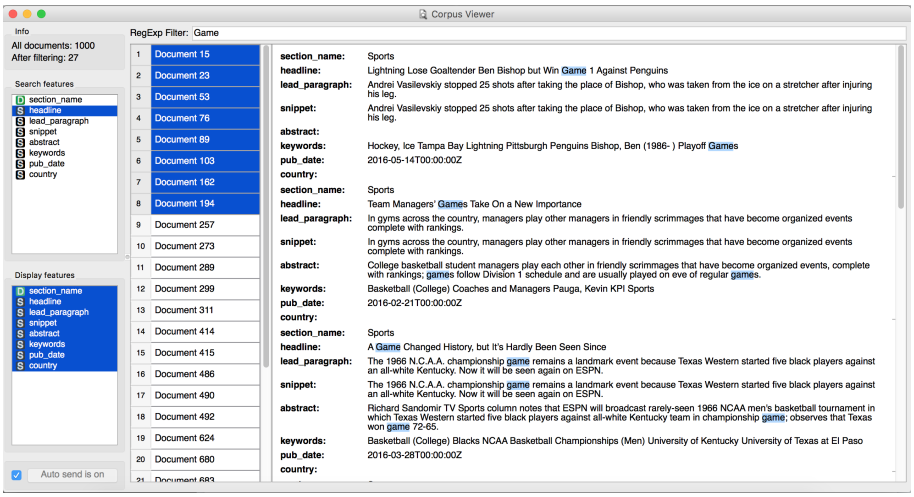
## 4.5 Vizualizacija

Orange že vsebuje bogat nabor vizualizacij, ki jih po gradnji ustreznih modelov, lahko uporabimo tudi na korpusih. Zato smo se odločili v našo knjižnico vključiti nekaj novih vizualizacij, ki so bolj tipične za besedila.

### 4.5.1 Corpus Viewer

Gradnik *Corpus Viewer* je brskalnik in pregledovalnik korpusov, ki hkrati omogoča tudi filtriranje dokumentov. Prikaz in filtriranje v gradniku nadzorujemo preko korpusovih atributov. Atributi izbrani za prikaz, določajo,

[13] `https://radimrehurek.com/gensim/models/ldamodel.html`

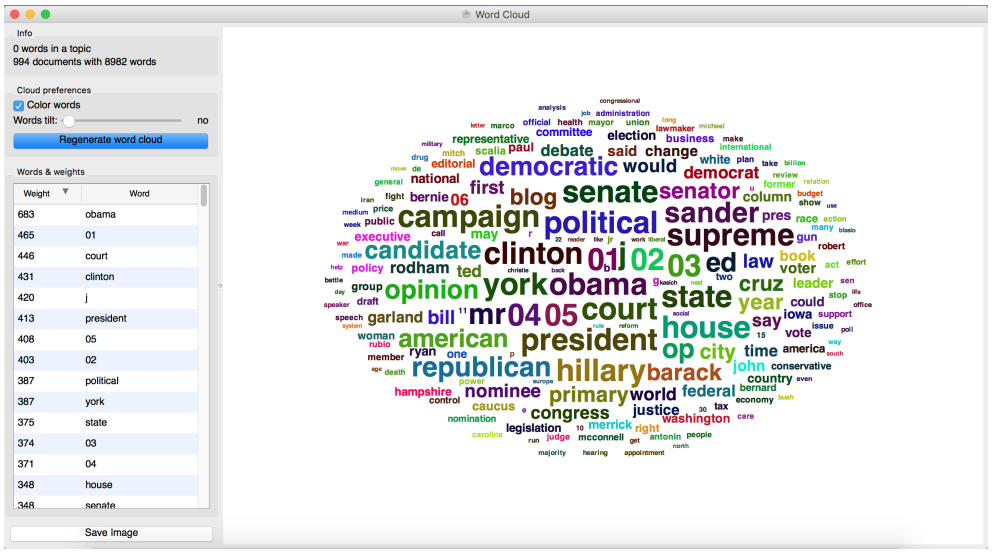



Slika 4.11: Na sliki vidimo korpus člankov s portala New York Times, odprtem v brskalniku *Corpus Viewer*. Ob skrajni levi strani okna so seznami atributov, kjer lahko izbiramo atribute za iskanje in atribute za prikaz dokumentov. V tem primeru smo v naslovncah iskali besedo *"Game"*, ustrezni dokumenti pa so izpisani v seznamu.

kako je sestavljen prikaz dokumentov. Če v pregledovalnik naložimo korpus New York Times člankov in kot prikazni atribut izberemo samo naslovnico, bomo v seznamu dokumentov videli samo naslovnice. Atributi izbrani za filtriranje, določajo po katerih atributih bomo filtrirali. Če nas zanimajo samo članki, ki v naslovnici vsebujejo besedo "Game", potem bomo za filtrirni atribut izbrali samo naslovnico (slika 4.11). Filtriranje dokumentov je tako bistveno hitrejše, ker program ne išče po celotnem dokumentu. Filtrirni kriterij določimo z vnosom besedne zveze oziroma regularnega izraza. Za branje regularnih izrazov smo uporabili Pythonov modul *re*. Dokumente, ki kriteriju ustrezajo (vsebujejo naveden izraz) lahko pošljemo na izhod za nadaljno uporabo v shemi, ostale pa gradnik filtrira.

### 4.5.2 Word Cloud

Oblak besed je ena izmed vrst vizualizacij, karakterističnih za besedilne podatke. Koncept vizualizacije je dokaj preprost, ampak je za uporabnika lahko




Slika 4.12: Na sliki je prikazana vizualizacija z oblakom besed v Orange. Vizualizacija prikazuje najpogostejše besede za New York Times članke o politiki. Z uporabo oblaka besed, lahko hitro opazimo, da med množico besed prevladujejo imena.

zelo informativen. Ideja je prikazati vse besede v množici eno ob drugi (sestavimo oblak) in jih poudariti sorazmerno s pogostostjo njihove pojavitve. To dosežemo tako, da bolj pogoste besede prikažemo večje ali pa jih pobarvamo z ustrezno barvo. Tipično je tudi, da izstopajoče besede postavimo bolj proti središču oblaka. Oblak besed v Orange lahko sestavimo z gradnikom *Word Cloud*. Ta na vhodu sprejme korpus dokumentov, lahko pa mu podamo tudi teme generirane z gradnikom za odkrivanje tem. Podobno kot pri grajenju vreče besed, oblaka ne moremo ustvariti, če dokumentov prej ne razčlenimo na besede. Gradnik zato žetone pridobi s klicem metode *tokens*, ki dokumente po potrebi razčleni.

Uteži za posamezne besede smo določili z uporabo razreda *Counter* Python modula *Collections*. Te so v gradniku prikazane v seznamu ob pripadajočih besedah, ki jih lahko tudi označimo. Besede v oblaku lahko prikažemo v barvah, določimo pa jim lahko tudi nagib (slika 4.12). Gradnik omogoča tudi shranjevanje slike v formatu *.png*. Prikaz besed je osnovan na PyQt gradniku

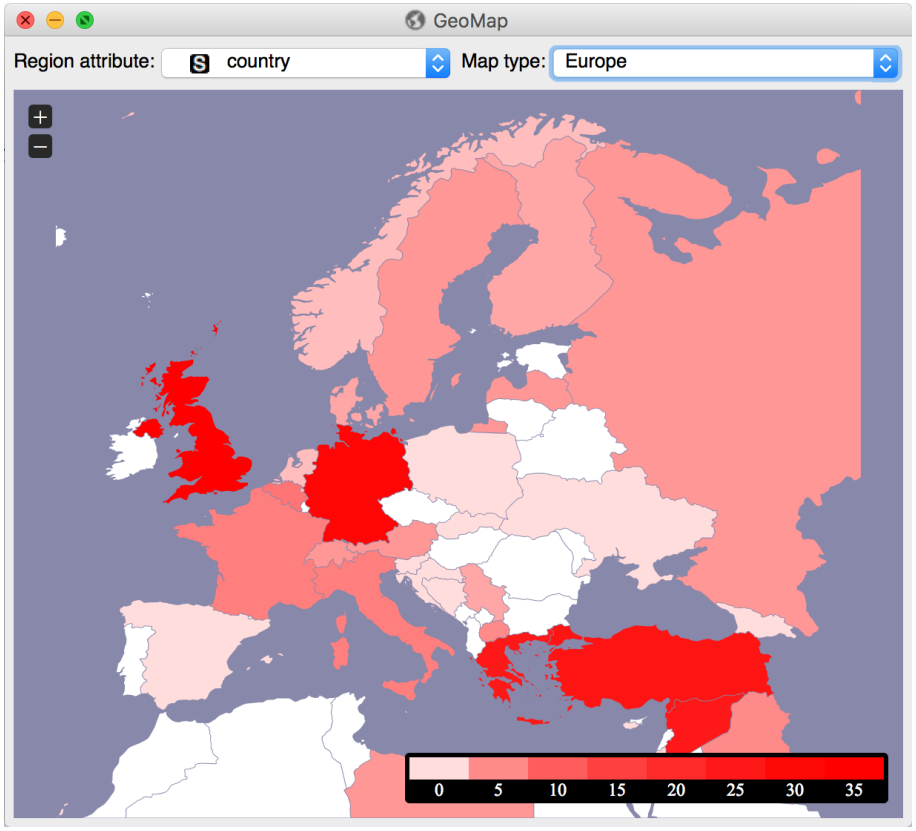
QWebView, ki je v Orange prirejen tako, da ne shranjuje zgodovine in lahko prikazuje podatke z lokalnih URL naslovov. Slika oblaka je vpeta v kodo HTML, animacije generiranja oblaka in nagibanja besed pa so definirane v skriptah napisanih v jeziku Javascript.

### 4.5.3 Geomap

Vizualizacija *Geomap* na zemljevidu sveta, prikaže kako pogoste so geografske entitete omenjene v dokumentih. Ideja za tako vrsto vizualizacije, izhaja iz dejstva, da ima večina današnjih objav na spletu, k vsebini pripete nekakšne geolokacijske metapodatke. Če pogledamo bolj praktičen primer, imajo članki pridobljeni z gradnikom *New York Times* med atributi polje, kamor se shrani tudi podatek o geografski lokaciji, ki je povezana z vsebino (na primer ime mesta ali države). Twitter celo nudi API vmesnik, za pridobivanje objav o lokacijah[14].

Zemljevid vizualizacije *Geomap* in pripadajoče animacije, so definirane s spletnima tehnologijama HTML in Javascript. Na zemljevidu lahko prikažemo celoten svet, lahko pa se omejimo na Evropo oziroma Združene države Amerike. Gradnik sam poišče ustrezen atribut, ki vsebuje geolokacijske podatke. V primeru, da jih je na voljo več, je mogoče tudi izbiranje preko spustnega menija. Ker so na zemljevidu označene samo države (slika 4.13), *Geomap* vrednosti, ki predstavljajo mesta in druge manjše entitete, po določenih pravilih pripiše ustrezni državi.

[14] `https://dev.twitter.com/rest/public/search-by-place`




Slika 4.13: Vizualizacija *Geomap*, prikazuje pogostost omemb evropskih držav v New York Times člankih, ki ustrezajo ključni besedi *"migration"*. Večkrat, ko je država omenjena, z bolj močno barvo je predstavljena.

# Poglavje 5

# Primeri uporabe

V tem poglavju so zbrani trije primeri uporabe Orange knjižnice za analizo besedil. Vključeni so primeri vizualnega programiranja in uporabe skriptnega dela knjižnice.

## 5.1 Ocenjevanje uspešnosti klasifikacije

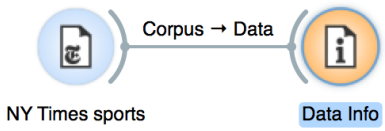
V grafičnem vmesniku Orange bomo zgradili shemo, s katero bomo ocenili učinkovitost uvrščanja člankov New York Times. Pri tem bomo primerjali dva pristopa. V prvem bomo klasifikacijo izvedli na podlagi frekvenc besed, v drugem pa bomo za klasifikacijo uporabili uteži tem, ki jih bomo odkrili z gradnikom *Topic Discovery*.

### Priprava podatkov

Kot vhodne podatke bomo uporabili članke z novičarskega portala New York Times, ki jih bomo zajeli z uporabo Orange gradnika *New York Times*. Ker bomo ocenjevali uspešnost klasifikacije, moramo poskrbeti za ustreznost podatkov:

- Podatki morajo imeti ustrezno število razredov med katerimi bomo uvrščali. Preveč razredov lahko klasifikacijo naredi težavno, po drugi strani, pa uvrščanje med malo razredi prinese manj informacije.

- Za vsak razred moramo imeti dovolj primerov. V nasprotnem primeru uvrstitve za te primere, ne bodo tako dobre, kot bi lahko bile sicer.

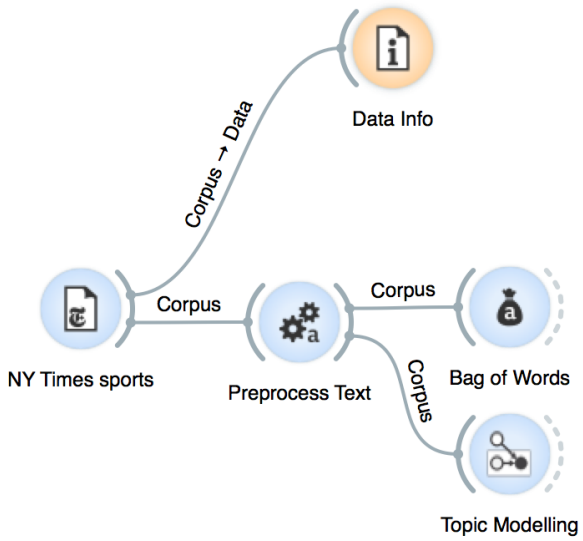



Slika 5.1: Podatke shranjene v tabeli, lahko preberemo z uporabo gradnika *Data Info*. Ta prikaže podatke o številu različnih razredov, atributov in meta atributov, velikosti tabele in lokaciji kjer je shranjena.

Izbrati moramo torej take ključne besede, ki nam bodo v skladu s tema ugotovitvama, vrnile najbolj ustrezne članke. O vsebini korpusa lahko v Orange poizvemo z uporabo gradnika *Data Info* (slika 5.1) Ta izpiše osnovne informacije, vključno s številom različnih razredov. Za ta primer, bomo zajeli članke objavljene med prvim januarjem in koncem maja leta 2016. Poizvedbe bomo opisali s ključnimi besedami *"business"*, *"technology"* in *"sports"* in na podlagi pridobljenih člankov ustvarili tri tematsko različne korpuse. Če po zajemu te korpuse primerjamo, ugotovimo, da smo za tematiko športa zajeli članke, ki pripadajo 24 različnim razredom, medtem ko članki o poslovanju pripadajo 25, članki o tehnologiji pa 27 različnim razredom[1]. Za korpuse, ki štejejo okoli tisoč dokumentov, je 24 različnih razredov popolnoma dovolj, zato bomo izbrali korpus s članki o športu.

Preden lahko pričnemo z ocenjevanjem uspešnosti klasifikacijskih metod, moramo podatke ustrezno predobdelati. Za to korpus s članki povežemo v gradnik *Preprocess Text* (slika 5.2). Pri razčlenjevanju bomo ločila izločili (uporabili bomo nastavitev *"NLTK tokenizer (no punctuation)"*). Žetone bomo krnili z metodo Porter, odstranili pa bomo tudi neinformativne besede angleškega jezika (vir besed *"English stop words"*). Vse velike črke bomo spremenili v male (vključili bomo nastavitev *"Case folding"*). Korpus

[1]Ta statistika velja za te konkretne poizvedbe, ki smo jih izvedli v tem primeru. Število razredov sicer variira glede na parametre poizvedb.




Slika 5.2: Na sliki vidimo korake, ki v Orange vmesniku za vizualno programiranje pokrijejo zajem, predobdelavo člankov in odkrivanje tem v njih.

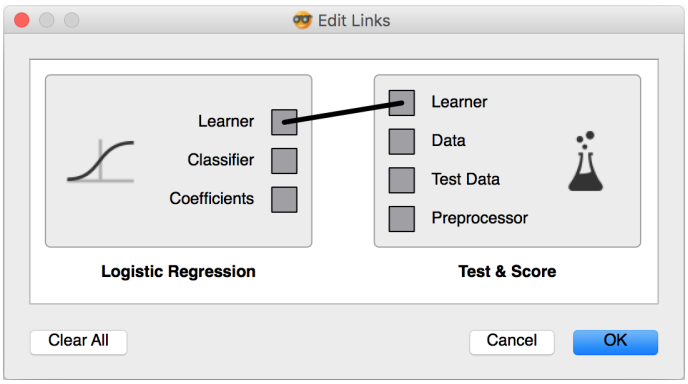
z žetoni bomo nato povezali v gradnik *Bag Of Words* in ustvarili model vreče besed, kjer bomo upoštevali mero TF-IDF.

Če želimo članke uvrščati na podlagi odkritih tem, moramo poiskati nekaj najpogostejših. To storimo tako, da predobdelan korpus povežemo z gradnikom *Topic Discovery*. Želimo, da bi bile odkrite teme kar najbolj informativne, hkrati pa ne želimo, da bi jih bilo premalo. Tako bomo nastavitev za število tem, nastavili na pet.

### Ocenjevanje uspešnosti klasifikacije

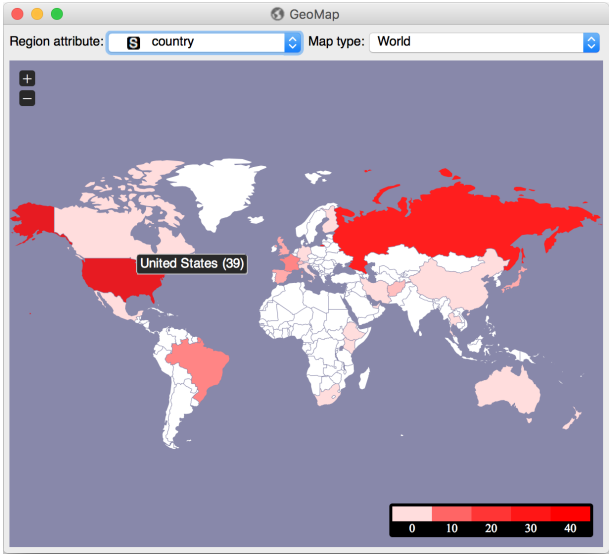
Po tem, ko smo zgradili vrečo besed in seznam uteženih tem, lahko začnemo z ocenjevanjem. To bomo opravili z uporabo Orange gradnika *Test & Score*, za metodo učenja pa bomo izbrali logistično regresijo. Ta je na voljo preko gradnika *Logistic Regression*, uporabili pa bomo privzete nastavitve (tip regularizacije *"Ridge (L2)"* in $C = 1$). Morda se zdi nenavadno, da smo za klasifikacijo elementov večrazredne množice izbrali binarni klasifikator, vendar s tem ni nič narobe. Logistična regresija v Orange v primeru večrazrednih

podatkov za klasifikacijo uporabi pristop eden proti ostalim (angl. *one vs. rest*). V tem procesu za $n$ razredov, naučimo $n$ klasifikatorjev. Ko gradimo klasifikator za nek razred, so primeri, ki mu pripadajo pozitivni, ostali pa negativni. Regresijo bomo v *Test & Score* podali na vhod *"Learner"*, na vhod *"Data"* pa bomo podali vrečo besed oziroma seznam uteži. Pri tem moramo biti pozorni na pravilnost povezav. Te lahko urejamo v oknu *"Edit Links"*, ki je dostopno z dvojnim klikom na povezavo med gradnikoma. Primer povezave lahko vidimo na sliki 5.3.




Slika 5.3: Ko povezujemo gradnike v Orange, moramo biti pozorni, da izhode povežemo z ustreznimi vhodi. Tu lahko vidimo urejanje povezave med gradnikom za logistično regresijo in gradnikom za ocenjevanje klasifikacije.

Za ocenjevanje rezultatov bomo uporabili osnovno prečno preverjanje (angl. *cross validation*) in prečno preverjanje tipa *Leave one out*. Naključno vzorčenje (angl. *random sampling*) bomo izpustili. Glede na število dokumentov in razredov je velika možnost, da v pridobljenem korpusu obstajajo razredi z enim samim primerom, medtem, ko naključno vzorčenje za vsak razred zahteva vsaj dva. Ta problem bi sicer lahko obšli z uporabo gradnika *Select rows*. Ta nam omogoča, da izberemo primere glede na specifičen kriterij, kar pomeni, da bi take primere lahko izpustili. Število podmnožic (angl. *number of folds*) prečnega preverjanja, bomo nastavili na 20, vključili pa bomo tudi nastavitev *"Stratified"*. Na ta način bodo podmnožice po sestavi podobne celoti. Kot ciljni razred (nastavitev *"Target class"*) bomo izbrali



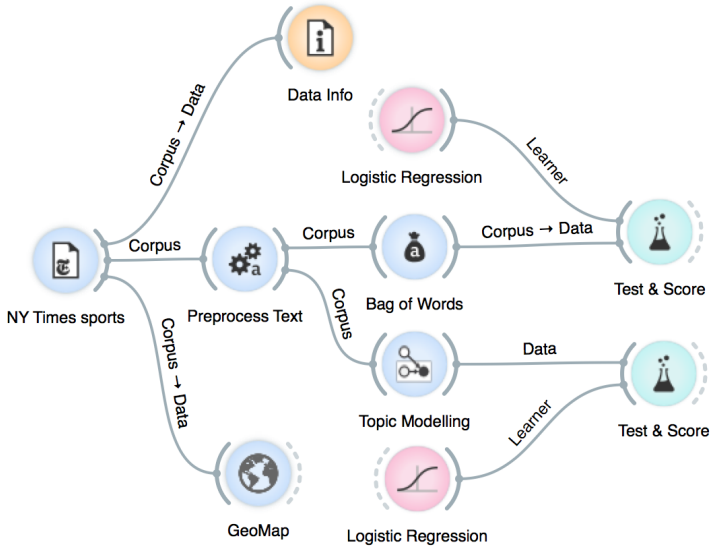

Slika 5.4: Na sliki vidimo vizualizacijo geografskih entitet v korpusu. Tu je prikazano kako pogosto se katera izmed držav pojavi v New York Times člankih o športu. Omembe večjih svetovnih mest, se preslikajo v omembe ustreznih držav.

razred "Sports".

Če sklepamo po rezutatih v tabeli 5.1, lahko zaključimo, da je uvrščanje z uporabo tem v primerjavi z frekvenco besed, vrnilo slabše rezultate. Teme so za uvrščanje morda vseeno preveč splošne. Rezultate bi lahko izboljšali na več načinov. Lahko bi povečali obseg korpusa in namesto tisoč člankov, uporabili na primer par tisoč člankov.

| | | natančnost | priklic |
|---|---|---|---|
| vreča besed | prečno preverjanje | 0.737 | 0.985 |
| | leave one out | 0.742 | 0.985 |
| teme | prečno preverjanje | 0.660 | 1.0 |
| | leave one out | 0.661 | 1.0 |

Tabela 5.1: V tabeli lahko vidimo rezultate uvrščanja člankov na podlagi frekvenc besed in uteži tem. Ocena postopka je podana z natančnostjo (angl. *precision*), ki nam pove kolikšen odstotek naših napovedi je bil pravilen in priklicom (angl. *recall*), ki nam pove koliko primerov tega razreda smo uspeli klasificirali.

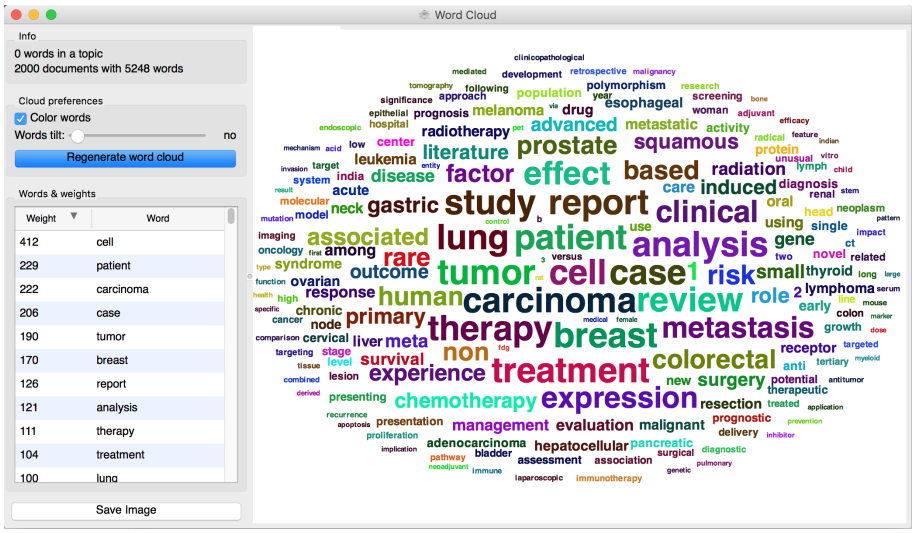



Slika 5.5: Na sliki vidimo celoten proces za ocenjevanje uspešnosti klasifikacije, ki smo ga sestavili v tem poglavju.

Tako bi posamezni razred imeli več primerov, kar bi izboljšalo kvaliteto učenja. Na korpus smo kot zanimivost pripeli tudi vizualizacijo *Geomap* (slika 5.4). Na njej lahko vidimo zemljevid sveta, kjer so države barvno označene glede na frekvenco omemb v člankih.

## 5.2 Uvrščanje člankov PubMed

V tem poglavju bomo demonstrirali, kako lahko z grafičnim vmesnikom Orange opravimo uvrščanje člankov zajetih z gradnikom *Pubmed*. Članke bomo po zajetju ustrezno predobdelali, nato pa iz njih ustvarili učno množico, na kateri bomo naučili klasifikator. Napovedovanje bomo izvedli z gradnikom *Predictions*, napačno uvrščene dokumente pa bomo nato pregledali v brskalniku *Corpus Viewer*.

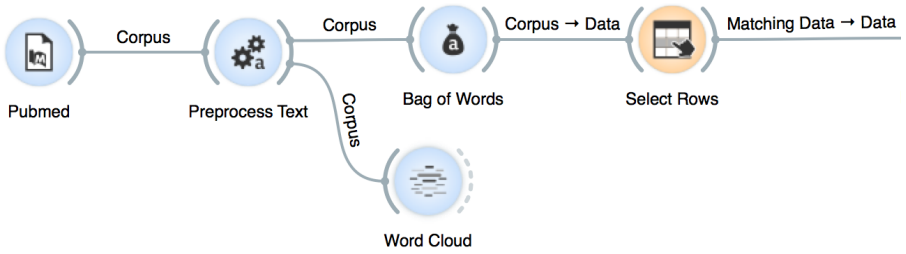



Slika 5.6: Vizualizacija z oblakom besed, ustvarjena z Orange gradnikom *Word Cloud*. Vizualizirane so besede v zbirki PubMed člankov o raku. Bolj karakteristične besede so postavljene v središče oblaka in so napisane z večjo pisavo. Besede smo zaradi lepšega prikaza lematizirali, namesto krnili.

### Priprava podatkov

Korpus objav bomo sestavili z uporabo gradnika *Pubmed*. Iskali bomo med članki izdanimi leta 2015. Kot ključno besedo bomo navedli *"cancer"*, iz člankov pa bomo izpustili meta atribut z informacijami o avtorjih. Čeprav so informacije o avtorjih lahko koristne, so nam pri nekaterih postopkih kot je vizualizacija, lahko v napoto. Avtorjev je ponavadi veliko, zato ni nenavadno, če se določena imena pojavijo večkrat. Na račun tega lahko v vizualizaciji kot je oblak besed (slika 5.6), neko ime hitro postane karakterisično za določeno skupino dokumentov, kar pa mogoče ni informacija, ki jo iščemo.

Za predobdelavo korpusa uporabimo Orange gradnik *Preprocess Text*. Pri razčlenjevanju bomo odstranili ločila (nastavitev *"NLTK tokenizer (no punctuation)"*), odstranili pa bomo tudi angleške neinformativne besede (nastavitev *"English stop words"*) in vse besede, ki se pojavijo v več kot 30% dokumentov (vrednost *"max_df"* nastavimo na 0.3.). Čeprav je krnjenje besed hitrejše, bomo za ta primer uporabili lematizacijo. Tako bo vizualizacija z

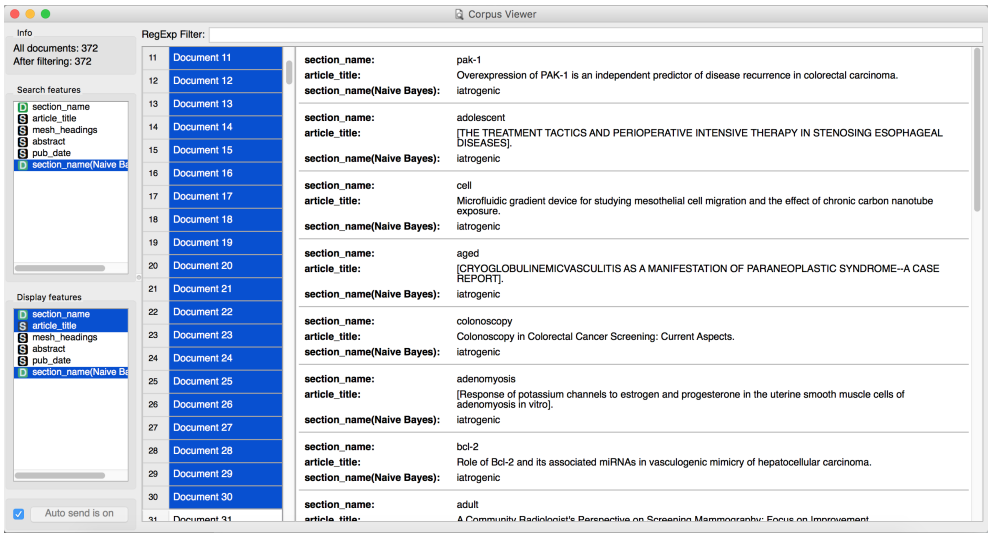



Slika 5.7: Filtriranje instanc v Orange lahko opravimo z gradnikom *Select Rows*. Tako lahko izločimo vse instance, ki ne pripadajo nobenemu izmed razredov.

oblakom besed bolj estetska, ker bo sestavljena iz celih besed[2]. Kot smo opisali v poglavju 4.2.3, se razredi, ki jim objave pripadajo, določajo na podlagi vrednosti polja MeSH. Namen tega polja je označevanje člankov, kar uporabnikom PubMed repozitorija poenostavi iskanje. Kljub temu pa nimamo nikakršnega zagotovila, da bo imela vsaka objava, ki jo pridobimo preko Biopython vmesnika Entrez, to polje izpolnjeno. Zato lahko računamo na to, da bo kar nekaj pridobljenih objav brez ustreznega razreda. To v Orange lahko rešimo na dva načina. Z gradnikom *Impute*, lahko manjkajoče vrednosti posameznih atributov in razredov, zamenjamo po nekem pravilu (na primer, jih nastavimo na nič ali pa na najbolj pogosto vrednost).

Drugi način, ki pa ga bomo uporabili v tem primeru je, da z gradnikom *Select Rows* izberemo samo tiste dokumente, ki imajo definiran razred (slika 5.7). Zato bomo preventivno zajeli več člankov (v tem primeru 2000), in potem filtrirali tiste, ki so brez razredov. Za to bomo v *Select Rows* ustvarili pravilo in za atribut *"section_name"* definirali pogoj *"is defined"*. Obkljukali bomo tudi nastavitvi *"Remove unused features"* in *"Remove unused classes"*, da odstranimo atribute in razrede, ki bodo po filtriranju odveč. V našem primeru, nam po filtriranju ostane še 1241 dokumentov. Odstranili smo take, ki so bili po zajemu brez razreda.

[2]Algoritem za krnjenje, bi besedam odstranil končnice.

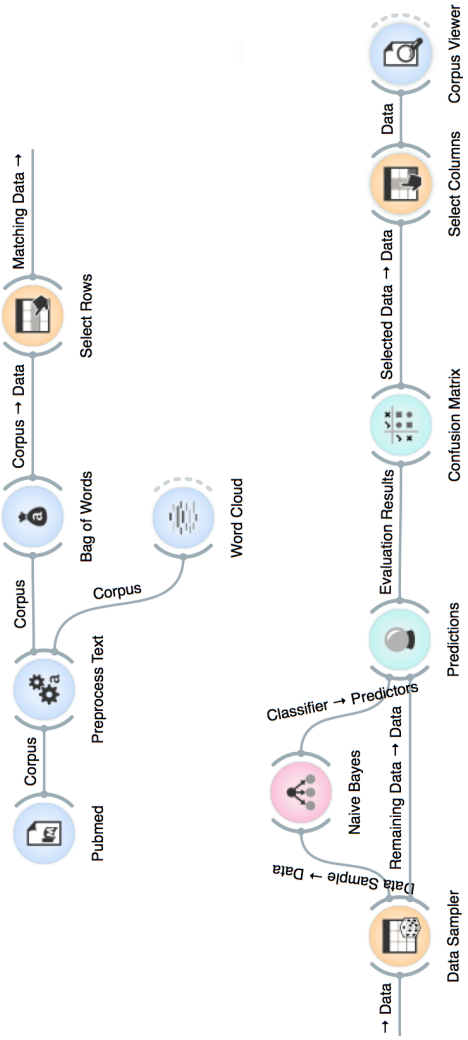



Slika 5.8: Napačno uvrščene dokumente lahko podamo pregledovalniku *Corpus Viewer*, kjer potem primerjamo vrednosti napovedanih in dejanskih razredov. Tako si lahko pomagamo pri identificiranju razlogov za napačno uvrščanje.

### Učenje klasifikatorja in napovedovanje

Predobdelan korpus z gradnikom *Data Sampler* razdelimo na učno in testno množico v razmerju 70 proti 30. Učna množica je tako na voljo na izhodu *"Data Sample"* in jo povežemo z gradnikom za učenje, ki bo v tem primeru *Logistic Regression*. Za napovedovanje uporabimo gradnik *Predictions*. Na njegov vhod *"Predictors"* povežemo Naivni Bayesov klasifikator naučen na naši učni množici, na vhod *"Data"* pa testno množico (povezava *"Remaining Data"*) iz gradnika *Data Sampler*. Primer te sheme lahko vidimo na sliki 5.9. Napovedi, ki jih vrne *Predictions* potem podamo gradniku *Confusion Matrix*, ki iz njih zgradi matriko pravilno in napačno uvrščenih objav. Tam lahko izberemo napačno uvrščene objave in jih pošljemo na izhod, kamor povežemo pregledovalnik *Corpus Viewer*. V pregledovalniku lahko pregledamo te dokumente in primerjamo njihove napovedane in dejanske razrede (slika 5.8).

Vidimo lahko, da je naš proces napačno uvrstil 372 objav, kar je približno




Slika 5.9: Na sliki je celoten proces za uvrščanje PubMed člankov, ki smo ga ustvarili v tem poglavju. Shema je zaradi obsežnosti predstavljena v dveh delih.

27% vseh. Če v pregledovalniku izberemo atribute *"section_name"*, *"section_name (Naive Bayes)"* in *"article_title"*, lahko primerjamo kam je Naivni Bayes objavo uvrstil in poskušamo najti korelacijo med naslovi člankov in rezultati uvrstitev. Kot iskalni atribut lahko izberemo napovedan razred in tako poiščemo dokumente, ki imajo nek specifičen napovedan razred.

## 5.3 Računanje razdalje med dokumenti s kompresijo

V tem poglavju bomo pokazali, kako lahko s skriptnim delom platforme Orange, opravimo uvrščanje člankov s stiskanjem besedil. V poglavju 2.1.4 smo opisali kako z uporabo kompresije določimo podobnost med dokumenti. Z uporabo tega koncepta bomo zdaj implementirali jedro in ga uporabili v metodi podpornih vektorjev.

### Priprava podatkov

Podatke bomo pridobili z uporabo razreda *NYT*. Zajeli bomo sto člankov s portala New York Times, izbrali pa bomo :

Izvorna koda 5.1: Zajeli bomo članke izdane med prvi januarjem in desetim junijem leta 2016, za vsako tematiko pa jih bomo zajeli sto.

```
api_key = 'api_key'
nyt_object = NYT(api_key)
sport = nyt_object.run_query(
        query='sport',
        date_from=date(2016, 1, 1),
        date_to=date(2016, 6, 10),
        max_records=100
)
sport.set_text_features(sport.domain.metas)
movies = nyt_object.run_query(
        query='movies',
        date_from=date(2016, 1, 1),
        date_to=date(2016, 6, 10),
        max_records=100
)
movies.set_text_features(movies.domain.metas)
```

Z metodo *set_text_features*, na korpusu določimo katere meta atribute želimo uporabiti pri generiranju dokumentov. Da bo uvrščanje bolj natančno želimo uporabiti vse, zato kot vhodni parameter v metodo podamo kar vse meta atribute, ki so na voljo v domeni korpusa (izvorna koda 5.1).

Ko zajamemo korpus New York Times člankov, zelo redko vsi vnosi padejo v en sam razred. Ker želimo uvrščati samo med dvema razredoma, bomo iz obeh korpusov izločili vnose, ki ne pripadajo razredu *"Sports"* oziroma *"Movies"*. Za to, bomo napisali pomožno metodo:

Izvorna koda 5.2: Z uporabo funkcije korpusove *from_corpus*, lahko ustvarimo nov korpus, ki vsebuje samo instance, ki jih specificiramo z ustreznimi indeksi.

```
def select_class(corpus, class_name):
    class_mapping = corpus.domain.class_var.values
    # Izberi indekse vnosov, z razredom "class_name".
    indices = [
        index
        for index, row in enumerate(corpus)
        if class_mapping[int(row.y[0])] == class_name
    ]
    # Izberi dokumente glede na izbrane indekse.
    return Corpus.from_corpus(corpus.domain, corpus, indices)
```

Preden lahko nadaljujemo, moramo nasloviti nekoliko specifičen problem. Implementacija metode podpornih vektorjev v Orange, temelji na implementaciji, ki jo najdemo v Python knjižnici *scikit-learn*[3]. Ta ne sprejema učnih podatkov v obliki nizov. To se morda ne zdi kot težava, saj besedila pred postopki učenja predobdelamo in prevedemo na model, ki uporablja vektorsko predstavitev s frekvencami. To sicer drži, vendar ne za pristop, ki ga želimo uporabiti v tem primeru. Jedro, ki ga bomo napisali, uporablja funkcijo za stikanje besedil, kar pomeni, da bodo podatki v obliki nizov, ki jih ne bomo predobdelali.

To omejitev bomo obšli z označevanjem besedil. Vsakemu besedilu bomo priredili neko zaporedno številko, in kot vhod v proces podali vektor teh števil. V jedru, bomo nato na podlagi teh števil lahko določili, kateri do-

[3] `http://scikit-learn.org/stable/`

kument moramo uporabiti. Kar potrebujemo, je torej preslikava med števili in dokumenti, za kar pa bo zadostoval kar običajen Python seznam. Preden sestavimo ta seznam, moramo sestaviti učno in testno množico dokumentov. Pri tem si bomo pomagali s sledečo pomožno metodo:

Izvorna koda 5.3: Korpus moramo razdeliti na učno in testno množico. Za to smo implementirali metodo *split_data*, ki korpus razdeli v razmerju 70 proti 30.

```
def split_data(corpus):
    documents = corpus.documents
    split_index = int(len(documents) * 0.7)
    train_set = documents[:split_index]
    test_set = documents[split_index:]
    return train_set, test_set
```

Kot lahko vidimo v primeru kode 5.3, nam pri definiranju množic ni potrebno vračati objektov tipa *Corpus*. Dovolj je, da vrnemo običajne Python sezname nizov. Naš referenčni seznam dokumentov želimo zgraditi tako, da bo prvih $n$ vnosov pripadalo učni množici, naslednjih $m$ pa testni. Tako bo dostopanje do dokumentov preko njihovih indeksov lažje:

Izvorna koda 5.4: Tu lahko vidimo, kako smo razdelili korpusa na učno in testno množico. V spremenljivko *document_mapping*, smo shranili vse dokumente, ki jih bomo uporabili v tem procesu.

```
# Korpuse razdelimo na ucno in testno mnozico.
sport_train, sport_test = split_data(
        select_class(sport, 'Sports')
)
movies_train, movies_test = split_data(
        select_class(movies, 'Movies')
)
# Sestavimo seznam za preslikavo dokumentov.
train_set = movies_train + sport_train
test_set = movies_test + sport_test
document_mapping = train_set + test_set
```

### Priprava klasifikatorja

Za učenje bomo uporabili Orange razred *SVMLearner*, za uvrščanje pa *SVMClassifier*. *SVMLearner* smo uporabili zato, ker za vhodni parameter sprejme

tudi implementacijo jedra. Tako *SVMLearner*, kot tudi *SVMClassifier*, smo nekoliko priredili:

Izvorna koda 5.5: Orange razred *SVMLearner* smo priredili tako, da smo določili predprocesorje in kot rezultat učenja, določili našo implementacijo razreda *SVMClassifier* (atribut *"__returns__"*).

```
class CompressionSVMClassifier(SVMClassifier):

    def __call__(self, data, ret=Model.ValueProbs):
        return super().__call__(data, ret)

class CompressionSVMLearner(SVMLearner):

    name = 'compression_svm'
    preprocessors = [RemoveNaNClasses(), RemoveNaNColumns()]
    __returns__ = CompressionSVMClassifier
```

Vhodni podatki v naš učni razred *CompressionSVMLearner* bodo samo indeksi dokumentov, zato nad njimi ne želimo izvajati predobdelave, kot je na primer normalizacija. Zato smo v naši implementaciji *SVMLearner* uporabili samo predprocesorje, ki odstranijo manjkajoče vrednosti.

Izvorna koda 5.6: Naše jedro za vhodna korpusa izračuna matriko razdalj. Preden nad dokumenti lahko izvedemo kompresijo, jih moramo pretvoriti v zaporedje bajtov. Dokumente dobimo preko njihovih indeksov v seznamu *document_mapping*.

```
def z(input_document):
    return len(compress(input_document))

def compression_kernel(corpus_a, corpus_b):
    # Za lazji dostop, se znebimo gnezdenih seznamov.
    indices_a = [int(doc[0]) for doc in corpus_a]
    indices_b = [int(doc[0]) for doc in corpus_b]
    # Ustvarimo prazno matriko za razdalje.
    distance_matrix = np.zeros(
            (len(indices_a), len(indices_b))
    )
    for mi_x, index_a in enumerate(indices_a):
        for mi_y, index_b in enumerate(indices_b):
            byte_a = bytes(document_mapping[index_a],
                           'UTF-8')
            byte_b = bytes(document_mapping[index_b],
                           'UTF-8')
```

```
18              # Izracunamo razdaljo med trenutnima dokumentoma.
19              distance_matrix[mi_x, mi_y] = -(
20                  (z(byte_a + byte_b) - z(byte_a)) / z(byte_a) +
21                  (z(byte_b + byte_a) - z(byte_b)) / z(byte_b)
22              )
23      return distance_matrix
```

Klasifikator *CompressionSVMClassifier*, ki smo ga osnovali na implementaciji razreda *SVMClassifier*, smo spremenili samo toliko, da nam bo poleg napovedi razredov, vračal tudi verjetnosti, ki jih izračuna za posamezne razrede. Jedro bomo definirali v lastni metodi in jo podali kot parameter *"kernel"* pri ustvarjanju objekta *CompressionSVMLearner*. Pri tem bomo definirali tudi metodo $z$, v katero bomo vključili stiskanje dokumentov (izvorna koda 5.6).

### Uvrščanje

Ker bomo v jedru do dokumentov dostopali preko njihovih indeksov v seznamu *document_mapping*, lahko kot vhod v instanco *CompressionSVMClassifier* uporabimo kar Orange tabelo. Ustvarili jo bomo iz matrike $X$, ki bo vsebovala indekse dokumentov učne množice in ustrezen vektor razrednih vrednosti $Y$. $X$ je pri več atributih matrika, v našem primeru pa bo vektor, saj imamo samo en atribut (indeks dokumenta). Ko naučimo klasifikator, lahko napovedovanje izvedemo na sledeč način:

Izvorna koda 5.7: Če želimo poleg napovedi dobiti tudi verjetnosti za posamezne razrede, potem parameter *"probability"*, nastavimo na *True*. Ko napovedujemo razrede, lahko namesto Orange tabele , podamo kar vektor indeksov, na katerih se v referenčnem seznamu nahajajo testni dokumenti.

```
1  # Ucenje.
2  learner = CompressionSVMLearner(
3          kernel=compression_kernel,
4          probability=True,
5          C=10.0,
6          gamma=0.01
7  )
8  # Napovedovanje.
9  classifier = learner(Table(X, Y))
10 predictions, probabilities = classifier(
11         np.array(range(
```

```
                len(train_set),
                len(document_mapping)
        ))[:, None]
)

print('Napovedi (movies):', predictions[:len(movies_test)])
# Napovedi (movies): [ 0.  0.  0. ... ]
print('Napovedi (sport):', predictions[len(movies_test):])
# Napovedi (sport): [ 1.  1.  1. ... ]
print('Verjetnosti:', probabilities)
# Verjetnosti: [[  9.97609143e-01   2.39085703e-03] ... ]
```

Napovedovanje s kompresijo se je v našem primeru obneslo zelo dobro. Od 53 testnih primerov, je algoritem vse uvrstil v ustrezen razred. Celotno kodo, ki smo jo uporabili za generiranje rezultatov, lahko vidimo v izseku kode 5.8.

Izvorna koda 5.8: V tem izseku je zbrana celotna koda, ki smo jo uporabili v tem primeru.

```
from datetime import date
from zlib import compress

import numpy as np
from Orange.base import Model
from Orange.classification import SVMLearner
from Orange.classification.svm import SVMClassifier
from Orange.data import Table
from Orange.preprocess import (RemoveNaNClasses,
                               RemoveNaNColumns)
from orangecontrib.text.corpus import Corpus
from orangecontrib.text.nyt import NYT

from svm_compression import CompressionSVMLearner


class CompressionSVMClassifier(SVMClassifier):

    def __call__(self, data, ret=Model.ValueProbs):
        return super().__call__(data, ret)


class CompressionSVMLearner(SVMLearner):
    name = 'compression_svm'
    preprocessors = [RemoveNaNClasses(), RemoveNaNColumns()]
    __returns__ = CompressionSVMClassifier

```

```
28
29 def select_class(corpus, class_name):
30     class_mapping = corpus.domain.class_var.values
31     # Izberi indekse vnosov, z razredom "class_name".
32     indices = [
33         index
34         for index, row in enumerate(corpus)
35         if class_mapping[int(row.y[0])] == class_name
36     ]
37     # Izberi dokumente glede na izbrane indekse.
38     return Corpus.from_corpus(corpus.domain, corpus, indices)
39
40
41 def split_data(corpus):
42     documents = corpus.documents
43     split_index = int(len(documents) * 0.7)
44     train_set = documents[:split_index]
45     test_set = documents[split_index:]
46     return train_set, test_set
47
48
49 def z(input_document):
50     return len(compress(input_document))
51
52
53 def compression_kernel(corpus_a, corpus_b):
54     # Za lazji dostop, se znebimo gnezdenih seznamov.
55     indices_a = [int(doc[0]) for doc in corpus_a]
56     indices_b = [int(doc[0]) for doc in corpus_b]
57     # Ustvarimo prazno matriko za razdalje.
58     distance_matrix = np.zeros(
59             (len(indices_a), len(indices_b))
60     )
61     for mi_x, index_a in enumerate(indices_a):
62         for mi_y, index_b in enumerate(indices_b):
63             byte_a = bytes(document_mapping[index_a],
64                            'UTF-8')
65             byte_b = bytes(document_mapping[index_b],
66                            'UTF-8')
67             # Izracunamo razdaljo med trenutnima dokumentoma.
68             distance_matrix[mi_x, mi_y] = -(
69                 (z(byte_a + byte_b) - z(byte_a)) / z(byte_a) +
70                 (z(byte_b + byte_a) - z(byte_b)) / z(byte_b)
71             )
72     return distance_matrix
73
74 api_key = 'api_key'
75 nyt_object = NYT(api_key)
76
```

```
sport = nyt_object.run_query(
        query='sport',
        date_from=date(2016, 1, 1),
        date_to=date(2016, 6, 10),
        max_records=100
)
sport.set_text_features(sport.domain.metas)

movies = nyt_object.run_query(
        query='movies',
        date_from=date(2016, 1, 1),
        date_to=date(2016, 6, 10),
        max_records=100
)
movies.set_text_features(movies.domain.metas)

sport_train, sport_test = split_data(
        select_class(sport, 'Sports')
)
movies_train, movies_test = split_data(
        select_class(movies, 'Movies')
)
# Sestavimo seznam za preslikavo dokumentov.
train_set = movies_train + sport_train
test_set = movies_test + sport_test
document_mapping = train_set + test_set

X = np.array(range(len(train_set)))[:, None]
Y = np.array(
        [0] * len(movies_train) + [1] * len(sport_train)
)[:, None]

learner = CompressionSVMLearner(  # Ucenje.
        kernel=compression_kernel,
        probability=True,
        C=10.0,
        gamma=0.01
)
classifier = learner(Table(X, Y))  # Napovedovanje.
predictions, probabilities = classifier(
        np.array(range(
                len(train_set),
                len(document_mapping)
        ))[:, None]
)
print('Napovedi (movies):', predictions[:len(movies_test)])
print('Napovedi (sport):', predictions[len(movies_test):])
print('Verjetnosti:', probabilities)
```

# Poglavje 6

# Zaključek

V okviru tega magistrskega dela smo s pregledom področja analize besedil identificirali njene temeljne postopke. Ugotovili smo, da analiza besedil izvira iz podatkovnega rudarjenja. Izpostavili smo glavno razlika med področjima, ki izvira iz lastnosti vhodnih podatkov. Besedila kot nestrukturirani podatki, zahtevajo poseben način predobdelave in prevedbo na ustrezen predstavitveni model. Pregledali smo tudi obstoječa orodja, ki rešujejo problem analize besedil. Orodja smo preizkusili in primerjali med seboj, na podlagi tega pa smo določili nabor funkcionalnosti, na katerih smo osnovali našo knjižnico.

Razvili smo knjižnico za analizo besedil v programskem okolju Orange. Njena implementacija se nahaja na spletnem repozitoriju GitHub[1] na naslovu `https://github.com/biolab/orange3-text` in obsega približno 6300 vrstic programske kode. Predpogoj za uporabo knjižnice je naložena delujoča verzija okolja Orange verzije 3. Navodila za instalacijo preko ukazne vrstice najdemo na spletnem naslovu `https://github.com/biolab/orange3`, lahko pa si naložimo tudi instalacijski paket, ki ga najdemo na naslovu `http://orange.biolab.si/download/`. Navodila o instalaciji Orange knjižnice za analizo besedil najdemo na spletnem naslovu `https://github.com/biolab/orange3-text`. Knjižnica je na voljo tudi preko sistema *pip*[2], naložimo pa jo lahko s sledečim ukazom:

[1] `https://github.com/`
[2] `https://pip.pypa.io/en/stable/`

```
$ pip install Orange3-Text
```

V knjižnico bi lahko dodali še podporo za spletne vire kot sta Twitter in Delicious. S Twitterja bi lahko pridobivali statusne objave uporabnikov in na podlagi tega odkrivali nove trende in jih prikazali na vizualizaciji s časovnico. Delicious je spletna stran za shranjevanje zaznamkov in je osnovana na družbene fenomenu imenovanem *social bookmarking* [22]. Z nje bi lahko pridobivali in analizirali najbolj priljubljene zaznamke. V modul za predobdelavo bi lahko vključili metode, kot je označevanje besednih vrst, posodobili pa bi lahko tudi sistem filtriranja žetonov z možnostmi filtriranja po njihovi dolžini, ali pripadnosti besednim vrstam. nabor vizualizacij, bi lahko dopolnili s prikazi kot so časovnice.

Knjižnica, ki smo jo razvili sledi smernicam učinkovitosti in preprostosti uporabe, ki jih Orange zastavlja. Nadaljnemu razvoju Orange bo sledil tudi nadaljni razvoj naše knjižnice. Z njeno implementacijo menimo, da smo dosegli cilj te magistrske naloge.

# Literatura


[1] I. H. Witten, Text mining, in: [23], 2004.

[2] A.-H. Tan, Text mining: The state of the art and the challenges, in: Proceedings of the Pacific Asia Conf on Knowledge Discovery and Data Mining PAKDD'99 workshop on Knowledge Discovery from Advanced Databases (KDAD'99), 1999, pp. 65–70.

[3] R. Feldman, J. Sanger, Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data, Cambridge University Press, New York, NY, USA, 2006.

[4] S. M. Stigler, Gauss and the invention of least squares, Ann. Statist. 9 (3) (1981) 465–474.

[5] R. A. Fisher, The use of multiple measurements in taxonomic problems, Annals of Eugenics 7 (7) (1936) 179–188.

[6] H. P. Luhn, A business intelligence system, IBM J. Res. Dev. 2 (4) (1958) 314–319.

[7] E. Brill, Some advances in transformation-based part-of-speech tagging, in: In Proceedings of the Twelfth National Conference on Artificial Intelligence, 1994, pp. 722–727.

[8] A. McCallum, D. Freitag, F. C. N. Pereira, Maximum entropy markov models for information extraction and segmentation, in: Proceedings of the Seventeenth International Conference on Machine Learning, ICML

'00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000, pp. 591–598.

[9] A. G. Zippo, Text classification with compression algorithms, CoRR abs/1210.7657.

[10] D. Benedetto, E. Caglioti, V. Loreto, Language trees and zipping, Physical Review Letters 88 (4) (2002) 48702.

[11] Measuring distance between unordered sets of different sizes, in: Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, 2014.

[12] J. Lewis, S. Ossowski, J. M. Hicks, M. Errami, H. R. Garner, Text similarity: an alternative way to search MEDLINE, Bioinformatics 22 (18) (2006) 2298–2304.

[13] M. D. Lee, B. Pincombe, M. Welsh, An Empirical Evaluation of Models of Text Document Similarity, Erlbaum, Mahwah, NJ, 2005, pp. 1254–1259.

[14] B. Li, L. Han, Distance Weighted Cosine Similarity Measure for Text Classification, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 611–618.

[15] D. A. Keim, Information visualization and visual data mining, IEEE Transactions on Visualization and Computer Graphics 8 (1) (2002) 1–8.

[16] J. Demsar, B. Zupan, Orange: Data mining fruitful and fun - A historical perspective, Informatica (Slovenia) 37 (1) (2013) 55–60.

[17] D. Flanagan, JavaScript: The Definitive Guide, Definitive Guides, O'Reilly Media, 2011.

[18] E. Loper, S. Bird, Nltk: The natural language toolkit, in: Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics -

Volume 1, ETMTNLP '02, Association for Computational Linguistics, Stroudsburg, PA, USA, 2002, pp. 63–70.

[19] R. Řehůřek, P. Sojka, Software Framework for Topic Modelling with Large Corpora, in: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, ELRA, Valletta, Malta, 2010, pp. 45–50.

[20] J. Demšar, T. Curk, A. Erjavec, Črt Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, B. Zupan, Orange: Data mining toolbox in python, Journal of Machine Learning Research 14 (2013) 2349–2353.

[21] D. M. Blei, A. Y. Ng, M. I. Jordan, Latent dirichlet allocation, J. Mach. Learn. Res. 3 (2003) 993–1022.

[22] M. G. Noll, C. Meinel, Web search personalization via social bookmarking and tagging, in: Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference, ISWC'07/ASWC'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 367–380.

[23] M. P. Singh (Ed.), Practical Handbook of Internet Computing, Chapman Hall & CRC Press, Baton Rouge, 2004.